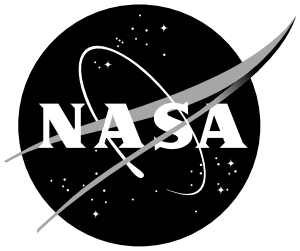


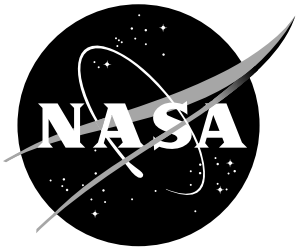
NASA Technical Memorandum 108517

Computer-Aided System Engineering and Analysis (CASE/A) Programmer's Manual, Version 5.0

J. Knox, Editor

September 1996

NASA Technical Memorandum 108517



Computer-Aided System Engineering and Analysis (CASE/A) Programmer's Manual, Version 5.0

J. Knox, Editor
Marshall Space Flight Center • MSFC, Alabama

National Aeronautics and Space Administration
Marshall Space Flight Center • MSFC, Alabama 35812

September 1996

TABLE OF CONTENTS

		Page
1.	INTRODUCTION	1
1.1	Basic Concepts and History	1
1.2	Program Description	2
1.3	Manual Organization	4
2.	SCHEMATIC MANAGEMENT	6
2.1	Model Creation, Loading, and Deletion	6
2.1.1	The Model (.MOD) File	7
2.1.2	Model Creation, Deletion, and Loading Routines	7
2.2	Schematic Manipulation	8
2.3	Component Manipulation	13
2.4	Connection/Icon Manipulation	16
2.5	Specialized Graphics Routines	17
3.	COMPONENT DATA BASE MANAGEMENT	21
3.1	File Input/Output Management	21
3.2	Interactive Editing	21
3.3	Solution System I/O	21
3.4	Data Management Framework	21
3.5	Data Management System Library Routines	25
3.5.1	File Input/Output Routines	25
3.5.2	Screen Editing Routines	27
3.5.3	Specialized Data Management Routines	27
4.	SIMULATION CONTROL AND EXECUTION	33
4.1	Execution Control Logic Description	33
4.1.1	Simulation Logic Segment Structure	33
4.1.2	Pseudo-Compute Sequence	36
4.1.3	User Operations Routines	36
4.2	Solution System Library Routines	37
5.	MODEL OUTPUT MANAGEMENT	41
5.1	Schematic Output	41
5.2	Component Data Base Output	41
5.3	Simulation Summary	41
5.4	Integrated Plot Utility	42
5.5	Data Output Options	43
5.5.1	Output to the “USERCON” Array	43
5.5.2	Custom User Output	44
5.5.2.1	Writing Data to an ASCII Text File	44
5.5.2.2	Creating Custom Data Base for Storing Output Data	45
5.6	Schematic Connection and Hydraulic Maps	47
5.6.1	The .CMP File	47
5.6.2	The .FMP File	49

TABLE OF CONTENTS

Page

6. UTILITY COMMANDS AND MISCELLANEOUS SYSTEM ROUTINES 51

6.1 Terminal Settings 51
6.1.1 Terminal Setting Command 51
6.2 Miscellaneous Commands 51
6.2.1 Flags 51
6.2.2 On-Line Help Information 52
6.2.3 VAX/VMS Commands 52
6.2.4 Temporary Exit to VAX Editor 53
6.3 Simulation Control Commands 53
6.3.1 Subsystem Heat Load Assignment to CABINS 53
6.3.1.1 ASSIGN Command 53
6.3.1.2 UNASSIGN Command 54
6.3.2 MERGE Operation 54
6.4 System Utility Routines 55
6.5 Model Archive Routines 61

7. USER OPERATIONS LOGIC AND INTERNAL CASE/A DATA ACCESS 63

7.1 OPS Logic Description 63
7.2 Creation of User OPS Logic 63
7.3 CASE/A Internal Data Communication Arrays 68
7.3.1 The “CON” Array 68
7.3.2 The “C” Array 69
7.3.3 The “PRO” Array 72
7.3.4 The “USERCON” Array 73
7.3.5 The “D” Array 73
7.3.6 Storage and Retrieval Functions for CASE/A Arrays 73

8. ANALYTICAL TECHNIQUES 76

8.1 System Pressure Computations 76
8.1.1 Matrix Reduction Pressure Solution 76
8.1.2 Hydraulic Solution 76
8.1.3 Stream Classifications 77
8.1.4 Friction Losses Through Connections 78
8.1.5 Pressure Loss Through Components 79
8.2 Thermal Network Solution Routines 80
8.3 Mass Transfer 81
8.4 Thermodynamic Properties 81

9. COMPONENT ROUTINES 82

9.1 Component Routine Logic Structure 82
9.1.1 Initialization Segment 83
9.1.2 Iterative Solution Segment 83
9.1.3 Posttime-Step Wrap-Up Segment 84
9.1.4 Postsimulation Wrap-Up Segment 84
9.1.5 Internal Fatal Error Condition 84
9.2 Component Routines 84

TABLE OF CONTENTS

Page

APPENDIX A. PREPARATION OF COMPATIBLE COMPONENT SUBROUTINES..... 95

1.0 Graphical Component Icon Construction..... 95
1.1 Step 1: Increase Number of Components..... 95
1.2 Step 2: Modify "Drawc" Routine 95
1.3 Step 3: Modify "Hit" Routine 99
1.4 Step 4: Modify "Locate" Routine 100
1.5 Step 5: Modify "Ssout" Routine 100
1.6 Step 6: Modify "Pinit" Routine..... 100
1.7 Step 7: Modify "Eqsolve" Routine..... 101
1.8 Step 8: Update CASE/A Object Library..... 101
2.0 Data Base Construction..... 103
2.1 Data Base File Description..... 103
2.1.1 Data Definition File 103
2.1.2 Binary Data File..... 103
2.1.3 Full Screen Editor Templates..... 103
2.1.4 Script File..... 104
2.2 Adding/Modifying Component Data Base Files..... 104
3.0 Component FORTRAN Routine 105
3.1 Data Initialization Segment..... 105
3.2 Iterative Solution Segment..... 106
3.3 Post-Time Step Wrap-Up Segment..... 106
3.4 Postsimulation Wrap-Up Segment..... 106
3.5 Example Component Subroutine–Heater 107

APPENDIX B. GLOSSARY OF LABELED COMMON BLOCK VARIABLES..... 111

APPENDIX C. THE MODEL (.MOD) FILE..... 119

APPENDIX D. INDEX OF CASE/A SUBROUTINES 123

LIST OF ILLUSTRATIONS

Figure	Title	Page
1.	CASE/A program sections	3
2.	Hierarchy of graphics routines	6
3.	SCREDT flow chart	22
4.	EDIT flow chart	23
5.	Example “DDF” file (CABIN data base)	24
6.	SOLVE routine flow chart	34
7.	Example connection map printout	37
8.	Variable output to the USERCON array	44
9.	Example PLOTSET definition for the USERCON array	44
10.	Writing custom data to a text file	45
11.	Example variable output using binary files	45
12.	Example code to get values for pump “P1”	46
13.	Example data definition file for custom output	46
14.	Example subsystem schematic	47
15.	Example connection map (.CMP file)	48
16.	Example hydraulic flow map (.FMP file)	50
17.	The “CON” array	70
18.	Stream properties and composition arrays	71
19.	Component “ITYPES”	82
20.	Component logic flow diagram	83
A-1.	Example code for new routine in Drawc routine	96
A-2.	MOLSIEV icon layout	96
A-3.	Example incon graphics code (MOLSIEV icon)	97
A-4.	Example hit box data initialization (MOLSIEV icon)	99
A-5.	Example hit stream label declaration (MOLSIEV icon)	100

LIST OF ILLUSTRATIONS (Continued)

Figure	Title	Page
A-6.	Modified library list (LIB4LIST.COM) ……	103
A-7.	Example data definition file ……	104
A-8.	Example source code for an ideal heater component……	107
C-1.	Example subsystem schematic……	119
C-2.	Example model description file (.MOD)……	120

LIST OF TABLES

Table	Title	Page
1.	Simulation control variables ……	35
2.	“C” array default constituents……	72
3.	“PRO” array properties list……	72
B-1.	CASE/A common blocks and included variables……	111

TECHNICAL MEMORANDUM

COMPUTER-AIDED SYSTEM ENGINEERING AND ANALYSIS PROGRAMMER'S MANUAL, VERSION 5.0

SECTION 1. INTRODUCTION

The Computer-Aided Systems Engineering and Analysis (CASE/A) modeling package was created as a generalized system engineering program. CASE/A is a very versatile and useful analytical tool, especially suited to the support of environmental control and life support system and active thermal control system (ECLSS/ATCS) development beginning with the system requirements preliminary design and proceeding through hardware development, integration, test, and operations. The basic architecture is very flexible and can be adapted to many other systems engineering problems in which the primary forcing functions are individual components. CASE/A can be used to verify component designs, examine critical operating conditions and parameters, establish system and subsystem performance, and determine test conditions and perform failure modes and effects analyses.

The CASE/A system has evolved from the G189A program[1] and shares much of the G189A program architecture. The CASE/A system is designed for applications in which G189A was used. Component routines from the G189A system have been used as models for many of the CASE/A components. CASE/A does not develop the solution in a traditional continuum mechanics manner similar to systems improve numerical differencing analyzer (SINDA) or finite element (FE) programs where the solution is derived by solving a set of differential equations. The CASE/A system solves the problem as a set of discrete point solutions, i.e., component by component. The approach is practical for systems where the primary drivers are the upstream/down stream conditions. The CASE/A system is designed to minimize the engineer/analyst's time and not necessarily the computer central processing unit (CPU) time. Attempts have been made to optimize the solution system, but not at the expense of the user. In today's world of rapidly advancing computer hardware, CASE/A is a logical approach to optimizing the system engineer's time.

It is recommended that both the User's Manual and the Programmer's Manual be kept readily accessible to facilitate learning the CASE/A system. The User's Manual is intended to guide the user in the operation of CASE/A, while the Programmer's Manual is intended to explain the program logic of CASE/A. Advanced users will find this document a useful reference in understanding the CASE/A system and in debugging CASE/A models. It is also highly recommended that experienced users locate the program files on their VAX system and type them to the terminal screen and follow along as they are discussed in this document.

1.1 Basic Concepts and History

The basic concepts fundamental to the CASE/A generalized ECLSS/ATCS analysis program include:

- A library of individual "component" subroutines, each of which is used to simulate a particular type of ECLSS/ATCS component or subsystem. The user specifies component performance parameters to tailor the component to his specific application.

- A data management framework that automatically manages all of the data associated with a particular ECLSS/ATC component and allows components to be grouped together and easily identified and manipulated.

- A solution system that controls the data transfer between components, provides an orderly solution of a set of component subroutines used to simulate a system, and allows component subroutines within the simulations to be easily replaced, modified, or interchanged.

- The capability of allowing the component subroutines to be interconnected with gas or liquid flow streams consisting of identifiable constituents.

- The use of a set of default parameters to simplify data entry.

CASE/A development has been funded under the NASA Marshall Space Flight Center (MSFC) contract NAS8-36407 and has been under continuous development since May 1985. The program structure was based largely on the G189A program with the addition of a graphical user interface for model construction and an improved data management system. The basic operating shell and component routine development work was completed in 1986. Overall program enhancements as well as additional component routine development work continued through 1987. A thorough component routine verification program was completed in early 1989, which consisted of validating and documenting atmosphere revitalization (AR) component model results against core module integration facility (CMIF) phase II comparative test results obtained from the MSFC space station technology test program. Recent enhancements to the user interface commands and program initialization code have further increased the engineer's productivity with CASE/A. The program currently contains approximately 70,000 lines of FORTRAN code. Version 5.0 is under MSFC Environmental Control and Life Support Branch configuration control.

1.2 Program Description

The CASE/A program provides a simulation tool for studying the transient performance of an ECLSS and/or an ATCS. CASE/A performs heat transfer, chemical reaction, mass/energy balance, and system pressure drop analysis based on user-specified operating conditions. A fluid system is simulated by connecting individual "components" through gaseous or liquid flow streams. Each particular type of component is simulated by an individual component subroutine. A component may represent a tee, a heat exchanger, a cold plate, or a process such as CO_2 removal or CO_2 reduction. The component subroutines are contained in a library that is accessible to the user. Each component is represented by a unique graphical icon. The user specifies the component interconnections by connecting the components interactively on the terminal screen. The syntax and user operations are explained in the User's Manual.

The CASE/A program consists of a main command processor that calls upon four primary sections as shown in figure 1. The user interfaces with the system through the command processor. The graphics management system enables the user to graphically create the system configuration to be modeled. Component, solution, and system data bases are maintained by the data base management system. The solution system controls the solution process and determines the calling order for the components. The component library is where the component object code resides.

The command processor/ graphical interface is described in the User's Manual. This is the top-level executive of the system and is the primary interface to the user. Essentially, the command processor parses the input commands into commands that can be executed by the system. The format of a command is:

Command;argument 1;argument 2.

Either a blank space or a semicolon can be used as a delimiter to separate the command and its arguments. Summaries of the commands and their syntax are provided in the User's Manual. The details of this system are described in section 2.

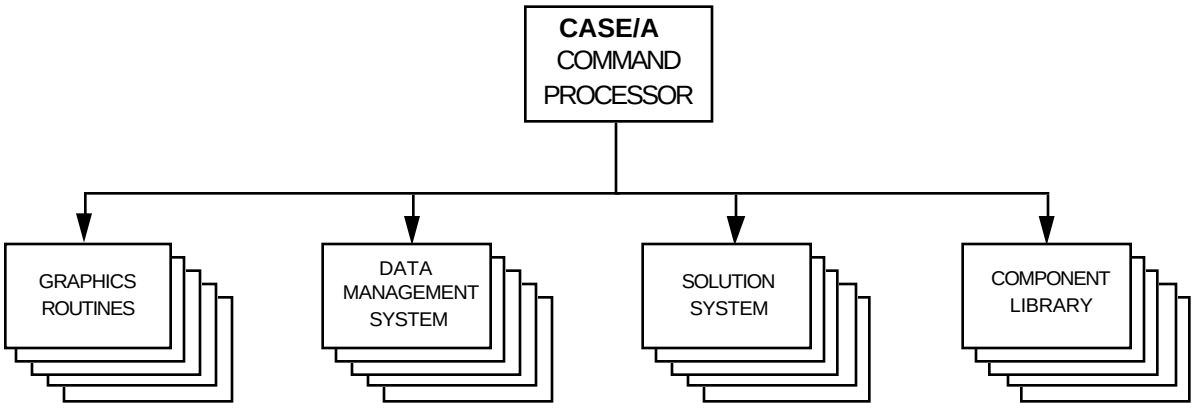


Figure 1. CASE/A program sections.

The data management system is based on the McDonnell Douglas Space Systems Company (MDSSC) technical data management system (TDMS). The data are stored in direct access FORTRAN files and are accessible only through the data management system or through FORTRAN programs. The description of the data base management system is provided in section 3. The data management system also provides the user with a data base management and plotting utility for manipulation and display of transient data.

The solution system controls the order of solution and provides for the system flags needed to determine the status of the system for the components. At present the solution order is sorted according to component type. The solution system is described in section 4.

CASE/A is an open system in that the user can create their own components and add them to the existing library. A generic blackbox component is provided for short-term applications where the user does not want to add the component to the system permanently. The components are allowed to have up to eight separate streams. The limitations are imposed by array dimensions, and some components (such as the cabin and store components) allow multiple connections to a single stream. Individual streams normally are expecting a specific constituent. For example, the Bosch component has a hydrogen inlet that expects hydrogen gas. The system does not check for incompatible connections such as connecting an air stream to a hydrogen stream. In such cases, diagnostic messages will be issued and the solution will attempt to proceed, but will generally not converge or will give erroneous results. All connections must be complete before requesting a solution or the system will terminate and provide error messages. The details of the component libraries and structure are discussed in section 9.

The CASE/A program has evolved from the G189A and SINDA programs. The architecture is very similar to G189A and the solution approach is similar to the approach used by SINDA. Thus, this system should be easy for the experienced G189A programmer to learn. Both CASE/A and G189A use component libraries and provide a component-by-component solution. The interface data for the components are passed through common blocks in both programs, and the component data are stored in a large array that is common blocked. A comparison of the primary arrays is presented below.

G189A	CASE/A	PURPOSE
R	CON	Stores component data
K/V	C and PRO	The C array contains the constituent data and the PRO array contains the property data. PRO array replaces locations 2 to 9 in K/V. C array locations are similar to location 1 and locations 10 to 19 in the K/V array.
A, B	None	The C and Pro arrays are also used as the working arrays, replacing the A and B arrays.

Because the model data are stored graphically, there is no direct comparison of the model data with G189A models. These data, which includes all components, connections, labels, notes, etc., are stored in the CASENAME.MOD file. A fundamental difference between CASE/A and G189 is the mechanism used to store the properties data. G189A uses the TABLE data which are input by the users in the model definition deck for every problem. CASE/A uses a subroutine called PROPS to calculate properties for the default constituents. This routine can be modified by the user to calculate properties for any desired constituent up to a maximum of 41 user-defined constituents. The program must be recompiled to incorporate the changes.

The solution system utilizes a pseudo-compute sequence (PCS) to control the solution order. The PCS concept is very similar to the concept used by SINDA to generate a solution. In SINDA, the node and conductor data are reduced to sequential ordered arrays that define the set of simultaneous equations. In CASE/A, the PCS is a map of the components and connections and is stored in the IPCS array. The solution routines march down the IPCS array and solve for each component individually.

1.3 Manual Organization

This section of the Programmer's Manual has provided a brief evolution and concept outline of the CASE/A system. Nine more sections, described below, are provided with the intent to familiarize the experienced CASE/A analyst with the programming concepts of CASE/A and to provide a foundation from which more detailed models can be developed. The basic structure of this document closely follows the structure of the CASE/A User's Manual.[2]

Section 2 discusses schematic management routines in CASE/A. Specifically discussed are those routines required to create, load, and delete models; those routines used to manipulate the graphic icons and connections representing a system; and specialized routines that perform rudimentary tasks such as drawing lines and circles.

Section 3 describes the data base management system used by CASE/A. Discussed here are routines that allow editing of data bases and routines used to transfer data between the data base files in centralized mass storage and CASE/A variables in memory.

Section 4 describes the routines used to control the simulation process. Discussed here are CASE/A and user-written routines and the logic of the simulation process.

Section 5 describes those routines used to manage the data processed by CASE/A. Described here are routines that output results in text or graphical form, and routines that output representations of a system such as connection maps and flow maps.

Section 6 discusses some of the utility commands such as terminal setting routines, on-line help, and interface routines to VMS, as well as those subroutines used by other routines to perform specific functions. Such routines perform functions such as interpolation, convergence checking, calculation of thermal properties, etc.

Section 7 discusses the user-supplied FORTRAN code referred to as OPS logic and how the user can access data in CASE/A internal storage arrays. OPS logic provides seven entry points at specified key simulation events where the user can intervene in the solution and customize the simulation. Stream property/constituent data and component data are maintained in the array data structures described in this section.

Section 8 provides details regarding the routines used to solve hydraulic, thermal, and mass transfer networks.

Section 9 provides a description of each component routine available in CASE/A.

Appendix A contains a description for adding new components to the CASE/A code.

Appendix B contains a table of the CASE/A global variables that are located in common blocks.

Appendix C contains a format description of the .MOD file that specifies components, connections, and other information for a model.

Appendix D contains an index of subroutines to help the reader locate a particular routine description in the manual.

References

1. "G-189A Generalized Environmental/Thermal Control and Life Support System Analysis Computer Program," 1977.

2. "Computer-Aided System Engineering and Analysis (CASE/A) User's Manual—Version 5.0," NASA TM–108514, June 1996.

VAX and VMS are trademarks of the Digital Corp. The names of other products and companies mentioned in this book may be trademarks, trade names, registered trademarks, service marks, or service names.

SECTION 2. SCHEMATIC MANAGEMENT

2.1 Model Creation, Loading, and Deletion

The graphical interface performs three major functions including:

1. Developing a graphical representation of the model under consideration

2. Interacting with a data base management system to store and manipulate equipment parameters

3. Storing the graphical representation in a format that is compatible with the solution routines.

The majority of the code in the graphical interface program is used to perform the first function listed above, developing the graphical representation. The data base management function relies heavily on the TDMS, which is described in section 3 of this document.

The graphical user interface is a graphics-based, command-driven package developed on the Digital Equipment Corporation (DEC) VAX minicomputer series and uses the Tektronix 4014-1™ terminal (or compatible) as the primary user input/output device. The interface uses a central command processor routine to accept user commands in text form and invoke the appropriate subroutines for action. The command processor separates the user input line into the command and optional arguments using the semicolon (;) or blank space as a general delimiter between fields.

The routines that comprise the graphical interface portion of CASE/A are broken down into distinct categories for discussion within this document. These categories include those routines pertaining to (1) schematic manipulation, (2) component manipulation, (3) connection manipulation, and (4) specialized graphics routines. These routines are shown hierarchically in figure 2. There are several routines that combine elements of the graphics/data management and graphics/solution system, and these routines will be discussed in this section of the document as well as their corresponding sections.

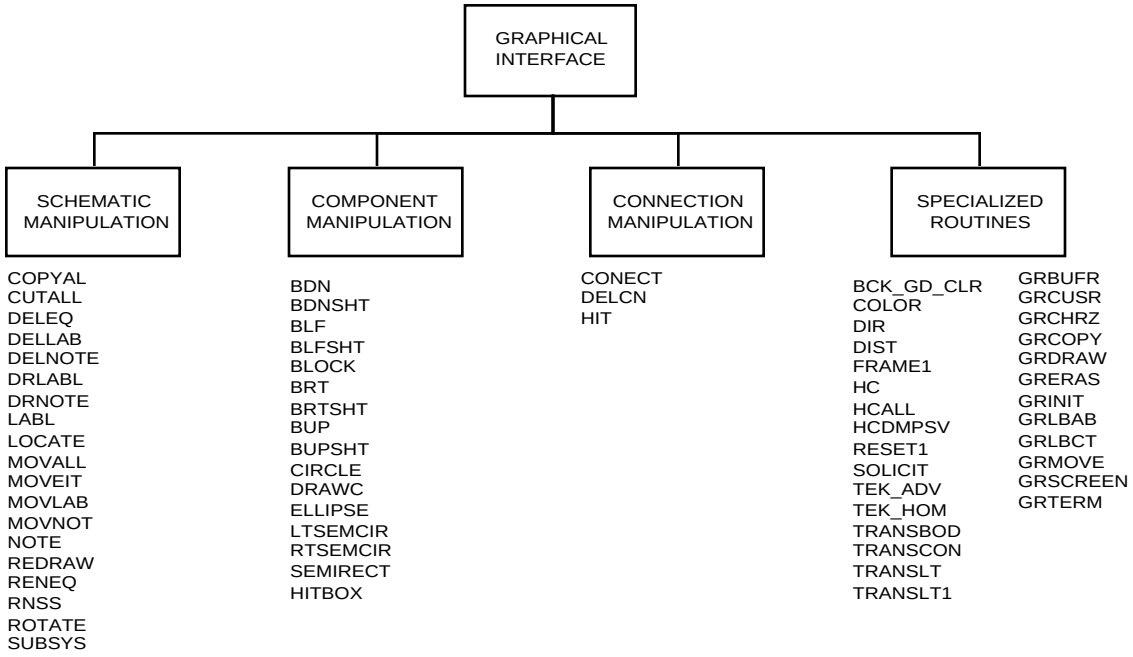


Figure 2. Hierarchy of graphics routines.

The following sections describe the routines associated with the development of the graphical representation of a given model. The actual model file CASENAME.MOD is discussed first. The subroutines that affect this file are presented next.

2.1.1 The Model (.MOD) File

All information necessary to create the subsystem screens is contained in the ".MOD" file. The first section contains model header data such as the number of components and connections in the model. This section serves as a header to describe the number of items contained in the following sections. The second section contains data describing each component in the model and its location on the schematic and assignment to a cabin. The third section contains information describing the connections between the components. The last two sections contain data describing user-defined blackbox components or notes data. The last two sections may not be present if the model does not contain these items. Refer to appendix C for a more detailed description of the ".MOD" file.

Since this file contains graphical information, it is highly recommended that this file not be modified through means other than the CASE/A interface (i.e., one should not use the VAX editor to modify this file). The MOD file is strictly formatted and contains a considerable amount of unlabeled data. As such, it is extremely difficult to debug a corrupted MOD file. Unlike other files associated with CASE/A (e.g., .BAK, .CMP, .FMP, .LPP), the MOD file MUST be present to run a given CASE/A model. All other files are created for the user by CASE/A. Unintentional editing errors introduced into a MOD file will corrupt a CASE/A model even to the point of rendering it useless. Should the ".MOD" file become corrupted, the user may be able to recover it from the ".BAK" file usually contained in the same directory.

2.1.2 Model Creation, Deletion, and Loading Routines

The following section describes the routines that, in conjunction with the data in the MOD file described above, are used to create or load a case into the CASE/A system.

Subroutine DELCAS

This routine invoked from CASEAMAIN. It deletes the current case from the data base records. It does NOT delete the files CASENAME.MOD, CASENAME.LPP, CASENAME.CMP, etc. After verifying the action, DELCAS opens each data base and finds a record with a case name match. The record is then deleted.

Functions and subroutines referenced:

DELREC EQOPEN FRAME1 RANDIN TEK_ADV

Subroutine DIR

This command routine allows the user to view a listing of all components in the case grouped according to subsystem name. The listing is directed to the terminal screen.

Functions and subroutines referenced:

CLS DIR_ADV FRAME1 PTMOD RJUSTIFY TEK_ADV TOGGLE_SCREEN

Subroutine CLOADCASE(NAME)

This routine reads the case data from the case definition file "CASENAME.MOD" and loads the corresponding data storage arrays for each data segment.

Functions and subroutines referenced:

CTOLOWERC CTOUPPERC DUPLICATE EQOPEN
FRAME1 RANDIN RANDOU
TEK_ADV

Subroutine CNEWCASE(NAME)

This routine is used to create a new case. It copies the default record for LABELS, CONTROL, PLOTSET and USERCON data bases to a new record with the new case name. It also initializes counters such as the number of cabins, connections, etc. Control is returned to the CASEA$MAIN upon completion.

Functions and subroutines referenced:

DUPLICATE EQOPEN FRAME1 RANDIN RANDOU
SAVE TEK_ADV

Subroutine SAVE

This command routine saves the current case configuration data in the case definition file "CASENAME.MOD".

Functions and subroutines referenced:

CTOLOWERC PTMOD TEK_ADV

Subroutine SAVEAS(NAME)

This routine creates a new case from the current case by duplicating all the data base entries with the new case name NAME. It also creates a new MOD file. The new case becomes the active case.

Functions and subroutines referenced:

DUPLICATE EQOPEN FRAME1 ISTAT MODBAK RANDIN
RANDOU SAVE TEK_ADV

2.2 Schematic Manipulation

The routines classified under this heading include all of those that pertain to the graphics setup on the working terminal screen. The functions provided by CASE/A to manage screen schematics include locating equipment; deleting equipment; adding or deleting labels and notes to a subsystem screen; moving equipment, notes, or labels to any location on a subsystem screen; and rotating equipment to another orientation in 90˚ intervals in order to make schematics more presentable. The syntax of the aforementioned operations are outlined in table 2.1-1 NASA TM–108514, "Computer-Aided System Engineering and Analysis (CASE/A) User's Manual—Version 5.0." A brief discussion of subroutines classified under this section, how they work, and the argument lists involved in their call statements follows.

Subroutine COPYALL

This routine invoked by the COPYALL command copies a portion (or all) of a subsystem screen to a new or existing subsystem by drawing a box around the components (500 maximum) to be copied. It includes notes, connections, and labels as well as components. Note that the blackbox component cannot be copied, and the copied items must be 30 pixels inside from the screen frame.

Functions and subroutines referenced:

CDCODE	CTOUPPERC	CILLCHAR	DUPLICATE
EQOPEN	FRAME1	GRALPH	GRCUSR
GRDRAW	GRMOVE	ISTAT	RANDIN
RANDOU	REDRAW	SAVE	TEK_ADV
TOGGLE_SCREEN			

Subroutine CUTALL

Deletes a portion (or all) of a subsystem screen by drawing a box around the components to be deleted. Notes, labels, and connections are also deleted.

Functions and subroutines referenced:

DELEQ	DELNOTE	GRALPH	GRCUSR
GRDRAW	GRMOVE	SAVE	TEK_ADV
TOGGLE_SCREEN			

Subroutine DELEQ(ICUT)

A command routine that allows the user to delete an existing component. The crosshairs are activated after the command is entered, and the component to be deleted is determined by locating the intersection of the crosshairs to within a maximum radius of 60 pixels to the center of that component. The equipment list (IEL) array, connection data (ICL) array, and component data base files are updated accordingly. All connections to the component are deleted, and the total number of connections (NCON) and total number of components (NEQ) are decremented.

Functions and subroutines referenced:

CDCODE	CTOUPPERC	DELREC	DIST
EQOPEN	GRCUSR	GRALPH	ISTAT
RANDIN	TEK_ADV	TOGGLE_SCREEN	

Subroutine DELLAB(ICUT)

A command routine that allows the user to delete an existing label from the subsystem screen. The crosshairs are activated after the command is entered, and the label to be deleted is determined by locating the crosshair coordinates to within a maximum radius of 60 pixels of the label center. The label data (ILB) array is updated accordingly, and the total number of labels (NLAB) is decremented.

Functions and subroutines referenced:

DIST	GRCUSR	TEK_ADV	TOGGLE_SCREEN

Subroutine DELNOTE(ICUT)

A command routine that allows the user to delete an existing note from the subsystem screen. The crosshairs are activated after the command is entered, and the note to be deleted is determined by locating the crosshair coordinates to within a maximum radius of 60 pixels of the note center. The note data (INOTE) array is updated accordingly, and the total number of notes (NNOTE) is decremented.

Functions and subroutines referenced:

DIST	GRCUSR	TEK_ADV	
TOGGLE_SCREEN			

Subroutine DRLABL (KLAB, MEQ)

A routine that draws the alphanumeric label and specified value for label number KLAB associated with relative equipment number MEQ. The labels data are contained within the array ILB.

Functions and subroutines referenced:

GRALPH	GRCHRZ	GRLBAB	KAM2AS
KAS2AM	SMVBITS	TEK_ADV	

Subroutine DRNOTE (K)

A routine that draws the alphanumeric note for note number K. The notes data are contained within the array INOTE.

Functions and subroutines referenced:

GRALPH	GRCHRZ	GRLBAB	GRLBCT
TEK_ADV			

Subroutine LABL

A command routine that allows the user to place an alphanumeric label that displays individual data base of stream characteristics for a given component on the subsystem screen. The labels data are contained in the array ILB, and the variable NLAB records the total number of labels in the case.

Functions and subroutines referenced:

CDCODE	DIST	DRLABL	HIT
GRALPH	GRCUSR	TEK_ADV	
TOGGLE_SCREEN			

Subroutine LOCATE

A command routine that allows the user to locate a new component in the active (currently displayed) subsystem. The equipment type and component name are command arguments. The crosshairs are activated after the command is entered and the component location is determined by the user. The component icon is then drawn on the screen at the indicated coordinates. The case equipment list is updated, the total number of equipment pieces (NEQ) is updated, and a new entry with default values is created in the component equipment data base file.

Functions and subroutines referenced:

CDCODE	CLS	CILLCHAR	CTOUPPERC
DRAWC	DUPLICATE	EQOPEN	GRALPH
GRCUSR	RANDIN	RANDOU	SAVE
TEK_ADV	TOGGLE_SCREEN		

Subroutine MOVALL

Moves a portion (or all) of a subsystem screen by drawing a box around the components (200 max) to be moved. Notes, labels, and the intermediate endpoints of connections inside the box (if either endpoint is on a component inside the box) are also moved.

Functions and subroutines referenced:

GRALPH	GRCUSR	GRDRAW	GRMOVE
SAVE	TEK_ADV	TOGGLE_SCREEN	

Subroutine MOVEIT

A command routine that allows the user to move an existing component to a new location on the current subsystem screen. The crosshairs are activated after the command is entered and the component to be moved is selected by the user. The crosshairs remain active, and the new location on the screen is designated by the user. The redraw command should be issued to clean the screen as the icon will not be erased from its previous position. The (IEL) array is updated to reflect the new component icon coordinates.

Functions and subroutines referenced:

DIST	DRAWC	GRALPH	GRCUSR
TEK_ADV			

Subroutine MOVLAB

A command routine that allows the user to move an existing label to a new location on the current subsystem screen. The crosshairs are activated when the command is entered and the label to be moved is selected by the user. The crosshairs remain active, and the new location on the screen is designated by the user. The redraw command should be issued to clean the screen as the icon will not be erased from its previous position. The ILB array is updated to reflect the new label coordinates.

Functions and subroutines referenced:

DIST	DRLABL	GRCUSR	TEK_ADV
TOGGLE_SCREEN			

Subroutine MOVNOTE

A command routine that allows the user to move an existing note from the subsystem screen. The crosshairs are activated after the command is entered and the note to be moved is selected by the user. The crosshairs remain active, and the new location on the screen is designated by the user. The redraw command should be issued to clean the screen as the icon will not be erased from its previous position. The note data (INOTE) array is updated accordingly.

Functions and subroutines referenced:

DIST	DRNOTE	GRCUSR	TEK_ADV
TOGGLE_SCREEN			

Subroutine NOTE

A command routine that allows the user to place an alphanumeric note on the active subsystem screen. The note character size, justification code, and character string are the command arguments. The crosshairs are activated when the command is entered and the note location is selected by the user. The note is then displayed on the screen at the indicated coordinates. The note data array (INOTE) is updated, and the total number of notes (NNOTE) is incremented.

Functions and subroutines referenced:

CDCODE	DRNOTE	GRCUSR	TEK_ADV
TOGGLE_SCREEN			

Subroutine REDRAW(JFLAG)

This subroutine will redraw the current subsystem screen.

Functions and subroutines referenced:

CLS	DRAWC	DRLABL	DRNOTE
FRAME1	GRDRAW	GRDRAWD	GRMOVE
RESET1	TRANSBOD	TRANSCON	TRANSLT1
TEK_HOM			

Subroutine RNSS

This subroutine allows the user to rename a subsystem.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CTOUPPERC	EQOPEN
RANDIN	RANDOU	TEK_ADV	

Subroutine ROTATE

A command routine that allows the user to change the orientation of a component on the current subsystem screen. The rotation angle is the command argument. The crosshairs are activated when the command is entered and the component to be rotated is selected by the user. The orientation of the component is then changed by “rotating” the component in a clockwise direction for positive rotation angle values relative to the existing orientation. The screen should be refreshed by the RD command since the previous orientation of the component will not be erased.

Functions and subroutines referenced:

CDCODE	DIST	DRAWC	GRCUSR
TEK_ADV	TOGGLE_SCREEN		

Subroutine SUBSYS

A command routine that allows the user to activate an existing subsystem or create a new subsystem. The subsystem name is input as the command argument, and the subsystem schematic is automatically drawn on the terminal screen.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CTOUPPERC	REDRAW
TEK_ADV			

2.3 Component Manipulation

The routines classified under this heading include those that are used to draw component icons (graphical representations of certain equipment types) on the subsystem screen. The CASE/A icon combines the equipment graphical representation, equipment name and type labels, and "hit boxes" to provide the complete component picture on the subsystem screen. The term "hit box" refers to a small square box of 10 by 10 pixels that is attached to the CASE/A component (one box per component stream) and allows the user to connect incoming and outgoing streams of successive components. Each component also has a "HIT DOT", used for the connection of controllers or timers. The component graphical representation and hit boxes are drawn with graphics routines from the TDMS library. At the present time, all component icons are limited to geometric shapes that can be formed from the following base structures; (1) circle, (2) left semicircle, (3) right semicircle, (4) square, and (5) straight line. The main driver routine for component icon creation is subroutine DRAWC. The DRAWC routine takes data points from the various component icon arrays (declared in DRAWC) and uses the graphics routines from the TDMS library to create a graphical representation of the component. A complete listing and description of these routines follows. Due to the complex nature of inserting a new component icon in the existing FORTRAN code, a step-by-step guide to this operation is included in appendix A.

Subroutine BDN (IX, IY, K, IROT,ISTM)

A routine that draws a "hit box" 10 pixels square with the center located at an X-coordinate of IX and Y-coordinate of IY-40. A line segment with a 35 pixel length is drawn from IY to IY-35 and has an arrow symbol drawn according to the code variable K. A value of K = –1 causes the arrow to point away from the box while a value of +1 causes the arrow to point toward the box. The variable (IROT) indicates the orientation angle of the component, and ISTM indicates the relative stream number of the component. This routine is used to draw the stream indicators on the component icons.

Functions and subroutines referenced:

BLF	BRT	BUP	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BDNSHT (IX, IY, K, IROT,ISTM)

Same as the BDN routine except the "hit box" is located 20 pixels instead of 40 pixels away from the (IX, IY) point.

Functions and subroutines referenced:

BLFSHT	BRTSHT	BUPSHT	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BLF (IX, IY, K, IROT,ISTM)

Same as the BDN routine except the “hit box” is located 40 pixels to the left of (IX, IY) point.

Functions and subroutines referenced:

BDN	BRT	BUP	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BLFSHT (IX, IY, K, IROT,ISTM)

Same as the BLF routine except the “hit box” is located 20 pixels instead of 40 pixels away from the (IX, IY) point.

Functions and subroutines referenced:

BDNSHT	BRTSHT	BUPSHT	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BLOCK (IXSIZ, IYSIZ, IX, IY, IROT)

A routine that draws a rectangle with X length of IXSIZ pixels and a Y length of IYSIZ pixels located around the centroid of point (IX, IY). The variable IROT indicates the orientation angle of the component.

Functions and subroutines referenced:

GRDRAW	GRMOVE

Subroutine BRT (IX, IY, K, IROT,ISTM)

Same as the BDN routine except the “hit box” is located 40 pixels to the right of the (IX, IY) point.

Functions and subroutines referenced:

BDN	BLF	BUP	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BRTSHT (IX, IY, K, IROT,ISTM)

Same as the BRT routine except the “hit box” is located 20 pixels instead of 40 pixels away from the (IX, IY) point.

Functions and subroutines referenced:

BDNSHT	BLFSHT	BUPSHT	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BUP (IX, IY, K, IROT,ISTM)

Same as the BDN routine except the “hit box” is located 40 pixels above the (IX, IY) point.

Functions and subroutines referenced:

BDN	BLF	BRT	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine BUPSHT (IX, IY, K, IROT,ISTM)

Same as the BUP routine except the “hit box” is located 20 pixels instead of 40 pixels away from the (IX, IY) point.

Functions and subroutines referenced:

BDNSHT	BLFSHT	BRTSHT	CIRCLE
GRDRAW	GRMOVE	HITBOX	

Subroutine CIRCLE (IRAD, IX, IY)

This routine draws a circle with radius IRAD centered at point (IX, IY) on the terminal screen.

Functions and subroutines referenced:

GRDRAW	GRMOVE

Subroutine DRAWC (KEQ)

This routine draws the graphical icon for the component with relative equipment number KEQ. The orientation of the icon is determined by the rotation angle contained in array element NROT (KEQ). This routine serves as the driver routine for icon representation and, hence, it uses many of the TDMS graphics library routines in order to draw the component graphically. A detailed description of how this routine must be modified in order to add new component icons to the existing list is found in appendix A.

Functions and subroutines referenced:

BDN	BLF	BLOCK	BRT
BUP	CIRCLE	COLOR	GRALPH
GRCHRZ	GRDRAW	GRLBAB	GRLBCT
GRMOVE	LTSEMCIR	RTSEMCIR	
SEMIRECT	TRANSLT		

Subroutine HITBOX(IX,IY,IXOFF,IYOFF,ISTM)

This routine draws the component icon hit box at a location of (IX+IXOFF, IY+IYOFF).

Functions and subroutines referenced:

BLOCK

Subroutine LTSEMCIR (IRAD, IX, IY, IOFF, IROT)

This routine draws a semicircle of radius IRAD located IOFF pixels along the X-axis from the coordinates (IX, IY). The positive clockwise rotation angle is specified by IROT.

Functions and subroutines referenced:

GRDRAW GRMOVE

Subroutine RTSEMCIR (IRAD, IX, IY, IOFF, IROT)

This routine draws a semicircle of radius IRAD located IOFF pixels along the X-axis from the coordinates (IX, IY). The positive clockwise rotation angle is specified by IROT.

Functions and subroutines referenced:

GRDRAW GRMOVE

Subroutine SEMIRECT(IH,IW,IX,IY,IROT)

This routine draws two parallel lines at IH/2 or IW/2 from (IX,IY) with a length of W or H depending on orientation. A rotation angle of 0 or 180 would result in horizontal lines.

Functions and subroutines referenced:

GRDRAW GRMOVE

2.4 Connection/Icon Manipulation

The connection management routines include those that affect component icon stream connections. CASE/A provides the user capability to add or delete a connection within the subsystem screen. The driver routines for the aforementioned commands are CONECT and DELCN, both of which call the subroutine HIT in order to verify a valid connection or deletion argument. The HIT routine checks to insure that the user has started and concluded the connect or delete process within the boundaries of a component hit box, otherwise the routine will not accept the user prompt and return control to the command processor. Following is a description of each routine and its arguments.

Subroutine CONECT

This command routine allows the user to specify an interconnect path between connection streams (a component such as a cabin must be connected to itself in the special case where heat transfer between the cabin and the surroundings is needed but there is no net flow into or out of the cabin). The crosshairs are activated when the command is entered and connections are resolved by determining which “hit boxes” of the target component(s) are specified by the user. The ICL array records, the connection data, and the total number of connections NCON are updated.

Functions and subroutines referenced:

CDCODE	GRALPH	GRCUSR	GRDRAW
GRDRAWD	GRMOVE	HIT	TEK_ADV
TOGGLE_SCREEN			

Subroutine DELCN

A command routine that allows the user to delete an existing interconnect path between component(s). The crosshairs are activated as the command is entered and the connection to be deleted is determined by locating the “hit boxes” indicated by the user. The (ICL) array is updated accordingly along with the total number of connections NCON.

Functions and subroutines referenced:

GRALPH GRCUSR HIT TEK_ADV
TOGGLE_SCREEN

Subroutine HIT (IX, IY, JEQ, NSTR,IXC,IYC)

This routine is used in the determination of interconnect paths between components. The routine determines the relative equipment number JEQ and stream NSTR, if any, which correspond to a "hit box" located at screen coordinates (IX, IY). A value of 0 for NSTR is returned if no "hit box" is located at the given set of coordinates. The active subsystem screen is the basis for the hit determination. The hit box data for each component is contained in arrays NST, IXD, and IYD. IXC and IYC are absolute locations of the hit box centers and are used by CONECT to draw connections on the subsystem screen.

Functions and subroutines referenced:

DIST RESET1 TRANSBOD TRANSCON
TRANSLT1

2.5 Specialized Graphics Routines

The routines listed as specialized graphics operations include those that can be invoked by the user to change or enhance the current screen characteristics. Many of these routines are called by routines whose categories fall in the above sections. For example, if the user entered the subsystem command in order to display a given subsystem, the specialized routine GRERAS (which erases the graphics window) would be called so that the new subsystem schematic would be drawn on a blank screen rather than over the existing screen. A complete description of all the routines categorized under this section heading follows. Low level graphics routines (with the prefix "GR") are part of TDMS and only their function is listed here.

Subroutine BCK_GD_CLR(IBACKGD)

This routine will set the background color index in the graphics mode. This is for Tektronix™ terminals only. Only solid colors are supported in this routine. The background color is specified in the HLS system, i.e., H-hue, L-lightness, S-saturation. Blue is 0,50,100. Colors are based on the factory defaults.

Functions and subroutines referenced:

KBIT

Subroutine COLOR(INDEX)

This routine will set the color index for the line in the graphics mode. This is for Tektronix™ terminals only.

Functions and subroutines referenced:

KBIT GRALPH

Subroutine FRAME1

This routine draws the borders and titles for the subsystem screen page layout. It is invoked on a redraw “RD”, redraw plus “RD+”, or subsystem “SS;name” command.

Functions and subroutines referenced:

CLDRAI	CLDRIA	GRALPH	GRCHRZ
GRERAS	GRLBAB	GRLBCT	GRMOVE
PTMOD	SYSCLK	TEK_HOM	

Subroutine GRALPH

This routine dumps the output buffer.

Subroutine GRCUSR (ICHAR, IX, IY)

This routine activates the graphics crosshairs so the cursor can be positioned by the user. The routine returns ICHAR, ADE (ASCII decimal equivalent) of the keyboard character depressed. It also returns the screen coordinates (IX, IY).

Subroutine GRCHRZ (ISIZ)

This routine sets the character size to a value of ISIZ, which can take on the values 1, 2, 3, or 4 with 1 being the largest and 4 the smallest.

Subroutine GRCOPY

This library routine generates a hard copy of the screen contents.

Subroutine GRDRAW (IX, IY)

This library routine draws a line from the present screen coordinates to the location of point (IX, IY).

Subroutine GRERAS

This routine erases the screen without changing the mode or current location of the cursor.

Subroutine GRINIT

This routine initializes the terminal and terminal status area.

Subroutine GRLBAB (IX, IY, ISTRING, LSTRING)

This routine displays an alphanumeric label string (ISTRING) of length (LSTRING) starting at the screen coordinates (IX, IY).

Subroutine GRLBCT (IX, IY, ISTRING, LSTRING)

This routine displays an alphanumeric label string (ISTRING) of length (LSTRING) centered about the screen coordinates (IX, IY).

Subroutine GRMOVE (IX, IY)

This routine moves the cursor from the current position to the screen coordinates (IX, IY).

Subroutine GRSCREEN(IWIDE)

This routine sets the screen width based on the value of IWIDE.

Subroutine RESET1 (IREC)

This routine sets the correct values in the memory arrays so that the component JEQ of equipment type KEQ is drawn in the desired orientation.

Functions and subroutines referenced: NONE

Subroutine TEK_ADV

This routine is used when the terminal is a Tektronix™. It is used to control the cursor location for overstrike terminals. It prevents overwriting command lines and graphics.

Functions and subroutines referenced:

FRAME1	GRALPH	GRCHRZ	GRERAS
GRMOVE	KBIT	REDRAW	TEK_HOM

Subroutine TEK_HOM

This routine homes the cursor on the Tektronix™ terminal.

Functions and subroutines referenced:

GRALPH	GRCHRZ	GRMOVE

Subroutine TRANSBOD (JEQ, IXC, IYC)

This routine returns the coordinates (IXC, IYC) of the control hit dot on the main body of the icon for equipment number JEQ based on an orientation angle of IRO that is held in the NROT(JEQ) array location.

Functions and subroutines referenced: NONE

Subroutine TRANSCON(JEQ, NSTR, IXC, IYC)

This routine returns the coordinates (IXC, IYC) of the control hit dot on the stream NSTR of the icon for equipment number JEQ based on an orientation angle.

Functions and subroutines referenced: NONE

Subroutine TRANSLT (IX, IY, IROT, IARRAY, JARRAY)

This routine adjusts the default orientation icon data contained in array IARRAY so the icon is drawn in orientation specified by angle IROT. The adjusted data are returned in the array JARRAY. The pivot coordinates are specified by (IX, IY).

Functions and subroutines referenced: NONE

Subroutine TRANSLT1(JEQ,KEQ)

This routine adjusts the default orientation of the component hit boxes.

Functions and subroutines referenced: NONE

SECTION 3. COMPONENT DATA BASE MANAGEMENT

The data management system is based on the McDonnell Douglas TDMS. This system is used to manage all of the CASE/A files including the component and system control files. Output files can be managed by the user within CASE/A in the integrated plot utility (IPU), both of which can be utilized for plotting and tabulating the results. For a complete description of the IPU see section 5.4 of the user's manual.

3.1 File Input/Output Management

The file I/O is managed through the primary routines: CEDIT, EQOPEN, and EQLOAD. The CEDIT routine is used in the interactive mode to display or modify the contents of the component, control, plot, label, or usercon data base parameters. The other routines are used to open a data base (EQOPEN), and load all of the data from a data base to the CASE/A system (EQLOAD).

3.2 Interactive Editing

The user can modify a data base parameter through the "ED" or "E" commands. The command processor will call the EDIT routine when either of the above commands is called. The "E" command (4014 graphics mode) allows the user to designate the component to be modified with the crosshairs. The "ED" command (VT100 text mode) will list all of the components of the type requested and solicit a choice from the user. If the user is working on a VT100 compatible terminal (determined by the variable ITMNL), the SCREDT (screen edit) subroutine will display the data base parameters on an edit screen and the user may change the data by simply typing over the old data. To advance to the next field, the user should press the return or the TAB key. This routine is depicted in figure 3. A flow chart of the EDIT routine is depicted in figure 4.

3.3 Solution System I/O

The solution system I/O is performed at the beginning and at the very end of the simulation to minimize disk I/O. All of the data for a given component are loaded and unloaded during these periods. Any modification to the CON array data must occur after the array is loaded. During simulation wrap-up, the data for each component is saved to the component data base files. During the save operation, the new component data will replace existing data for that component record.

3.4 Data Management Framework

The data for each component are stored in a direct access binary file by the name of COMPONENTNAME.DAT (PUMP.DAT, MOLSIEV.DAT, etc.). This file contains data according to the format specified by a "Data Definition File" COMPONENTNAME.DDF. These files typically reside in the [CASEA.CASEA_V5.DATA] centralized component data directory, which must be defined by the VMS LOGICAL NAME CASEA$DATA. A typical "DDF" is shown in the example in figure 5. The DDF begins with a data line (header record) that provides the following characteristics of the data base:

1. Record length in words
2. Number of items in each record (attributes)
3. Item location of the modification date item
4. Maximum number of records for this data base

5. Item location of the security password
6. Item location for the “point up” pointer for chained records (usually zero)
7. Item location for the “point down” pointer for chained records (usually zero).

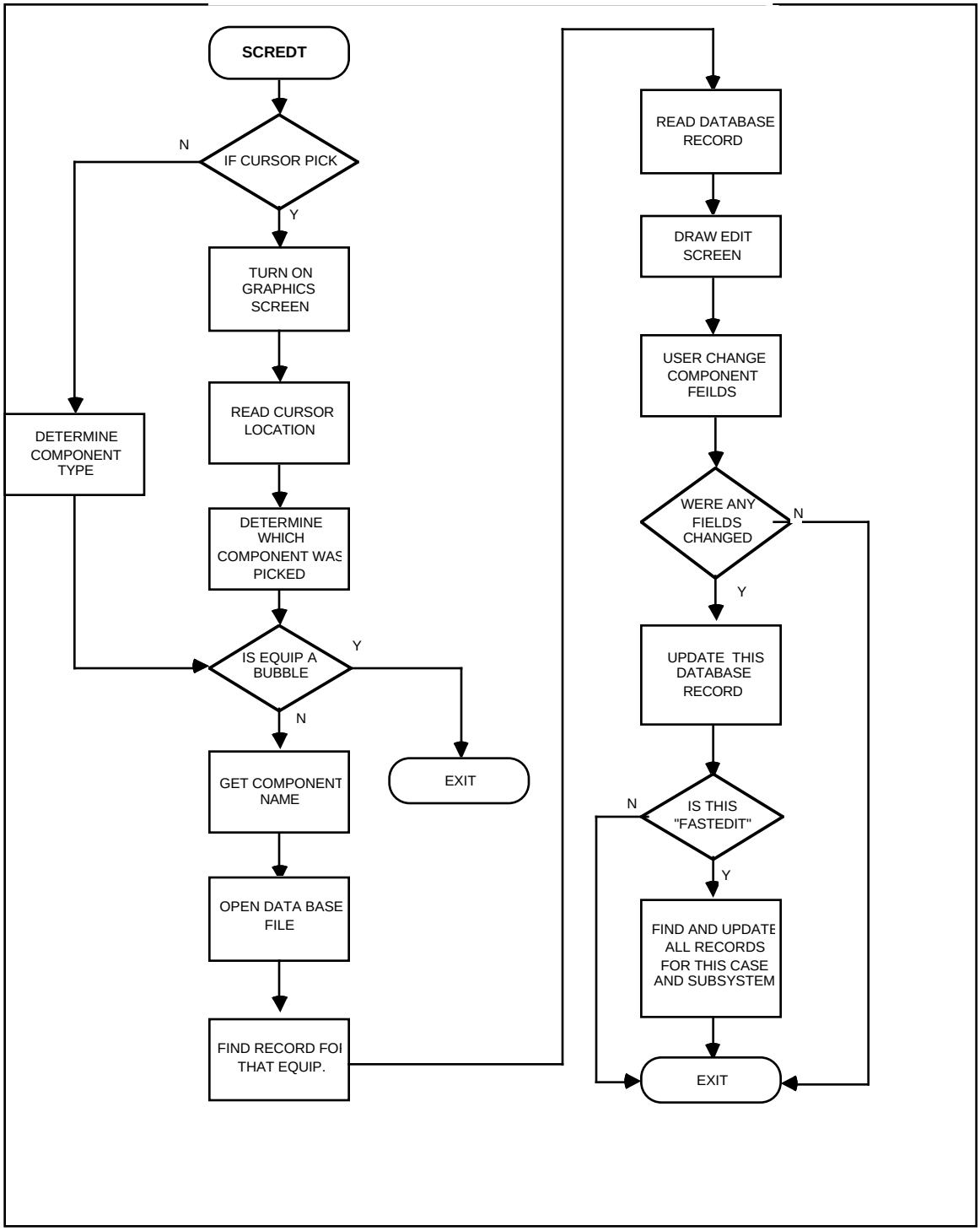


Figure 3. SCREDT flow chart.

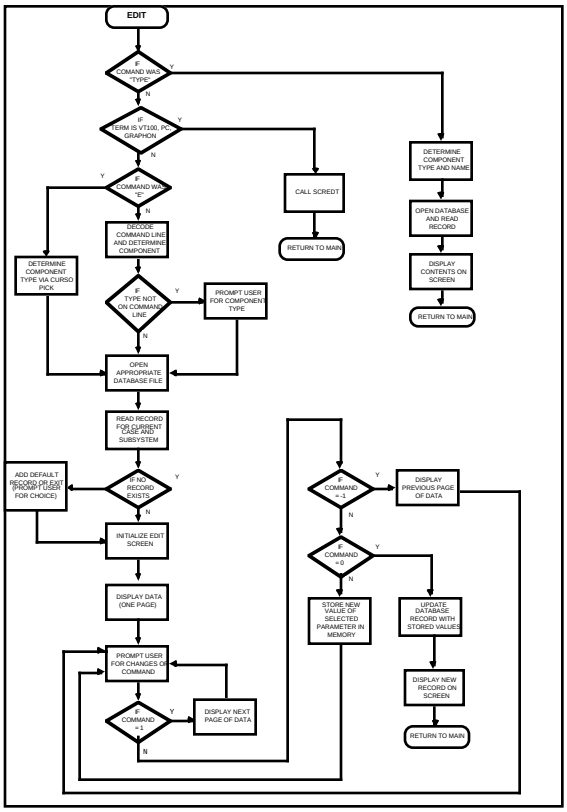


Figure 4. EDIT flow chart.

```
60 57 56 500 57 0 0                    ! DDF header record
002 002 001 CASE NAME                  ! 2  word alpha field starts in word 1
002 002 003 SUBSYSTEM NAME
002 002 005 COMPONENT NAME
003 001 007 NO. OF INPUTS              ! 1 word real field starts in word 7
003 001 008 NO. OF OUTPUTS
003 001 009 NO. OF BENCHMARKS
003 001 010 CDEL RXCC
003 001 011 TDEL RXCC
003 001 012 PDEL RXCC
003 001 013 POWER,WATTS
003 001 014 WEIGHT, LBM
003 001 015 VOLUME, FT3
003 001 016 CABIN LEAKAGE, LBM/DAY
003 001 017 EXPER VENT AIR LOSS, LBM/DAY
003 001 018 EVA AIR LOSS, LBM/DAY
003 001 019 CABIN VOLUME (CONSTANT), FT3
003 001 020 RELIEF VALVE SETPOINT PRESS,PSIA
003 001 021 INITIAL CABIN AIR TEMP, DEG-F
003 001 022 INITIAL TOTAL PRESS, PSIA
003 001 023 INITIAL O2 PARTIAL PRESS, PSIA
003 001 024 INITIAL CO2 PARTIAL PRESS, MMHG
003 001 025 INITIAL REL HUMIDITY, %
003 001 026 CABIN AIR HEAT LOAD, WATTS
003 001 027 INITIAL CABIN STRUCT TEMP,DEG-F
003 001 028 ENVIRONMENTAL SINK TEMP, DEG-F
003 001 029 INTERIOR CONVECTION, BTU/HR/F
003 001 030 SHELL CONDUCTANCE, BTU/HR/F
003 001 031 EXTERNAL RADK, BTU/HR/R**4
003 001 032 THERMAL RELAXATION COEFFICIENT
003 001 033 THERMAL MAX LOOP COUNT
003 001 034 STRUCTURE MASS, LBM
003 001 035 EXTERIOR CONVECTION, BTU/HR/F
003 001 036 FINAL CABIN TEMP, DEG F
003 001 037 FINAL TOTAL PRESS, PSIA
003 001 038 FINAL O2 PARTIAL PRESS, PSIA
003 001 039 FINAL CO2 PARTIAL PRESS, PSIA
003 001 040 FINAL H2O PARTIAL PRESS, PSIA
003 001 041 MAX TOTAL PRESS, PSIA
003 001 042 MIN TOTAL PRESS, PSIA
003 001 043 NOM TOTAL PRESS, PSIA
003 001 044 MAX O2 PARTIAL PRESS, PSIA
003 001 045 MIN O2 PARTIAL PRESS, PSIA
003 001 046 NOM O2 PARTIAL PRESS, PSIA
003 001 047 MAX CO2 PARTIAL PRESS, MM HG
003 001 048 MIN CO2 PARTIAL PRESS, MM HG
003 001 049 NOM CO2 PARTIAL PRESS, MM HG
003 001 050 MAX REL HUMIDITY, %
003 001 051 MIN REL HUMIDITY, %
003 001 052 NOM REL HUMIDITY, %
003 001 053 MAX DEW POINT, DEG F
003 001 054 MIN DEW POINT, DEG F
003 001 055 NOM DEW POINT, DEG F
003 001 056 MAX TEMPERATURE, F
003 001 057 MIN TEMPERATURE, F
003 001 058 NOM TEMPERATURE, F
001 001 059 MOD DATE
002 001 060 SECURITY
$$EX                                   ! Security ID
```

Figure 5. Example “DDF” file (CABIN data base).

This header record is followed by a data line for each of the items in the record. The format is as shown in the figure and contains the following:

1. Item type (1 = integer, 2 = alpha, 3 = real, 4 = bit flag, 5 = list)

2. Field size in words (Item types 1, 3, 4 and 5 are 1 word fields; item type 2 field size is the number of characters/4 rounded up to nearest whole word)

3. Starting word position in the record

4. Description of the item (32 characters max).

The bit flag items are supported by TDMS for purposes not required by CASE/A and, therefore, are beyond the scope of this manual. The DDF ends with a four character "security" ID that currently is designed to prevent inadvertent modification of protected records as opposed to protection against malicious modification. At present the security feature of the code has been disabled.

The third file used by the data management system is the "COMPONENTNAME.STT" file that contains the status records for the active set. The data bases are not automatically compressed when a record is deleted. The deleted record location is flagged as available for entering new data. Any new data will be entered over the old record and that record will be flagged as a part of the active set. These flags are in the .STT file, which is stored in binary form.

3.5 Data Management System Library Routines

The data management system library routines consist of a large number of routines that work together in order to maintain component, connection, and schematic data bases. The routines are grouped into file input/output, screen editing, and specialized data management routines.

3.5.1 File Input/Output Routines

The following routines provide functions to store/load data from disk to/from random access memory (RAM). The function of each routine is listed below along with the major arrays and variables that are updated inside the routine.

Subroutine CLOADCASE(NAME)

This routine reads the case data from the case definition file "CASENAME.MOD" and loads the corresponding data storage arrays for each data segment.

Functions and subroutines referenced:

CTOLOWERC	CTOUPPERC	DUPLICATE	EQOPEN
FRAME1	RANDIN	TEK_ADV	

Subroutine CMPOPEN

This routine writes a connection map of the pseudo-compute sequence of the current case into a file named "CASENAME.CMP" after the solve command has been issued.

Functions and subroutines referenced:

CTOLOWERC	SMVBITS

Subroutine EQLOAD

This routine loads component data into the “CON” array from appropriate data base records.

Functions and subroutines referenced:

DUPLICATE	EQOPEN	RANDIN	RANDOU
TEK_ADV			

Subroutine EQOPEN (ITYPE)

This routine opens the active data base file for equipment type ITYPE for interactive editing or listing to the terminal screen. Control, labels, plot, and usercon data bases have ITYPES of 0, -1, -2, and -3, respectively. Component ITYPES are listed in chapter 9.0.

Functions and subroutines referenced:

CDCODE	CTOLOWERC	FULL	RANDIN
TEK_ADV			

Subroutine ITMOUT(ITEM,IA,IOUT)

This subroutine determines the character format and writes specific items to the unit specified.

Functions and subroutines referenced:

CDCODE	FORMAT	KBIT	SETPRIM
TEK_ADV	TYCON		

Subroutine LISOPEN

This subroutine opens the case “.LPP” file.

Functions and subroutines referenced:

CTOLOWERC

Subroutine PTMOD

This subroutine loads the 12th column of the IEL array with assignment data if the component is in a subsystem that is assigned to a cabin.

Functions and subroutines referenced:

SMVBITS

Subroutine RANDIN(ILOCK,IUNIT,IREC,IA,NWD)

This routine is used to retrieve data from the CASE/A data bases. It reads the record number IREC from the file having logical file unit number IUNIT into the array IA with a record length of NWD. If an error occurs the IA array is set equal to zero.

Functions and subroutines referenced:

BLANK	TEK_ADV

Subroutine SAVE

This command routine saves the current case configuration data in the case definition file "CASENAME.MOD".

Functions and subroutines referenced:

CTOLOWERC PTMOD TEK_ADV

3.5.2 Screen Editing Routines

The following routines can be invoked by the user to modify an existing component data base or to edit any other file on the current disk storage device.

Subroutine CEDIT(IEDIT)

This command routine allows the user to modify or list the parameters for a desired component. The type of component to be edited is specified in the command argument list along with the optional component name. The data file COMPONENTYPE.DAT is opened and searched for the desired component. Once found, the component parameters are displayed on the screen and the user designates changes to be made. The contents are then updated, displayed to the screen again, and the data are saved to disk.

The value of IEDIT determines what kind of editing is done. Values of 0, 1, 2, or 3 give the possibility of a normal edit from command prompts, normal edit via cursor pick of component, global edit (FASTED) via cursor pick, or display of data base values with no edit, respectively.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CTOUPPERC	DIST
DUPLICATE	EQOPEN	FRAME1	GRCUSR
ISTAT	ITMOUT	RANDIN	RANDOU
SAVE	SCREDT	TEK_ADV	TYCON

Subroutine SCREDT(IEDIT)

SCREDT is called by EDIT if the terminal is a VT100 type text terminal. IEDIT determines the kind of editing to be performed as above.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CLS	CTOLOWERC
CTOUPPERC	DIST	DUPLICATE	EQOPEN
GRCUSR	GRSCREEN	ISTAT	LIST_EDITOR
RANDIN	RANDOU	RESTOR	SAVE
SCREEN_MANAGER		SETPRIM	
SUB_EDITOR	TEK_ADV		

3.5.3 Specialized Data Management Routines

The specialized data management routines are those that are used by the other routines in order to manipulate data into distinct formats. Some of the specialized routines correspond to the MDAC TDMS and others are VAX specific routines. A function of each routine in this section is listed along with any of the major arrays and variables that are affected inside the routine.

Subroutine CDCODE (CLINEJ, LOC, ITYP, IWORD, IDATA, IERR)

This routine is equivalent to DCODE but used with CHARACTER array of arbitrary length, CLINEJ. This routine is used to decode the alphanumeric text typed at the CASE/A prompt. It decodes the alphanumeric argument array ICOM into various fields delimited by the semicolon ";" or blank character. The argument LOC specifies the field number, ITYP specifies the data type to be returned, and IWORD specifies the length of the field in words. There are four possible values for ITYP; a value of 1 specifies an integer, a value of 2 specifies an alphanumeric, a value of 3 specifies a real number, and a value of 4 specifies the binary representation of the field. The decoded field value is returned in the argument IDATA. The argument IERR returns a value of 0 if an error occurred during the decode, otherwise a nonzero value is returned.

Functions and subroutines referenced:

CFIELD	CREADAL	CREADFL	CREADIN
INDEX	TEK_ADV		

Subroutine CFIELD (CLINEJ, LOC, ICOLL, ICOLH, IERR)

This is equivalent to FIELD but with CHARACTER array of arbitrary length, CLINEJ. This is an auxiliary routine to CDCODE. It finds the field number LOC delimited by semicolons or blanks in the argument array ICOM. ICOLL and ICOLH are used to keep track of the current position within the array ICOM. The argument IERR returns the error code 0 if no character in field number LOC is found or 1 if the operation is successful.

Functions and subroutines referenced:

TEK_ADV

Subroutine CLDRAI (NDATE, NYR, NMO, NDA)

This routine breaks the variable NDATE, which consists of a concatenated value for the date, into the corresponding year (NYR), month (NMO), and day (NDA).

Functions and subroutines referenced: NONE

Subroutine CLDRIA (NYR, NMO, NDA, NDATE)

This routine takes the integer values for the current year (NYR), month (NMO), and day (NDA) and returns the concatenated value (NDATE).

Functions and subroutines referenced: NONE

Subroutine CREADAL (CLINEJ, ICOLL, ICOLR, IDATA, IERR)

This routine returns the CHARACTER value in CLINEJ located between columns ICOLL (left) and ICOLR (right) in the IDATA variable.

Functions and subroutines referenced: NONE

Subroutine CREADALC (CLINEJ, ICOLL, ICOLR, IWORD, CDATA, IERR)

This routine returns the CHARACTER value in CLINEJ located between columns ICOLL (left) and ICOLR (right) in the CDATA variable.

Functions and subroutines referenced: NONE

Subroutine CREADFL (CLINEJ, ICOLL, ICOLR, DATA, IERR)

This routine returns the REAL value in CLINEJ located between columns ICOLL (left) and ICOLR (right) in the DATA variable.

Functions and subroutines referenced: NONE

Subroutine CREADIN (CLINEJ, ICOLL, ICOLR, IDATA, IERR)

This routine returns the INTEGER value in CLINEJ located between columns ICOLL (left) and ICOLR (right) in the IDATA variable.

Functions and subroutines referenced: NONE

Subroutine DELREC (IRECL, IRECH)

This routine deactivates the records IRECL through IRECH from the data base.

Functions and subroutines referenced:

DEL	LBIT	OPENDB_X	RANDIN
RANDOU	SETPRIM	TEK_ADV	

Subroutine FIELD (ICOM, LOC, ICOLL, ICOLH, IERR)

This is an auxiliary routine to DCODE. It finds the field number LOC delimited by semicolons or blanks in the argument array ICOM. ICOLL and ICOLH are used to keep track of the current position within the array ICOM. The argument IERR returns the error code 0 if no character in field number LOC is found or 1 if the operation is successful.

Functions and subroutines referenced:

RBYTE

Subroutine FILREC (IA, IDATA)

This routine fills the array IA with the alphanumeric data located inside the variable IDATA.

Functions and subroutines referenced:

SBYTE SMVBITS

Subroutine FINDRM (IREC)

This routine returns the first open record IREC in the data base parameter file.

Functions and subroutines referenced:

RANDIN SETPRIM

Subroutine ITEMIO (ITEM, IA)

This routine outputs the individual value in location ITEM of array IA. It is used for displaying the data base parameters for various components or control files when the editing routine is invoked by the user.

Functions and subroutines referenced:

FORMAT SETPRIM TYCON

Subroutine KAM2AS (NCHAR, KA4, KADE)

This routine converts the first NCHAR characters of the alphanumeric array KA4 into ADE (ASCII Decimal Equivalent) format in the array KADE.

Functions and subroutines referenced:

SMVBITS

Subroutine KAS2AM (NCHAR, KADE, KA4)

This routine converts the first NCHAR characters of array KADE originally in ADE format into alphanumeric format inside the array KA4.

Functions and subroutines referenced:

SMVBITS

Subroutine LBIT (NS, NL, IW, IV)

This is a bit manipulation routine that copies the leftmost NL bits from argument variable IV into the target variable IW starting at the NS bit location of IW.

Functions and subroutines referenced:

SMVBITS

Subroutine MVBITS (ISORC, ISTRT1, ILEN, IDEST, ISTRT2)

An intrinsic VAX subroutine that moves data bits from one location to another. The arguments are specified as follows.

ISORC – An integer variable or array element that contains the bits to be transferred.

ISTRT1 – An integer expression that identifies the position of the first bit within ISORC to be transferred.

ILEN – An integer expression that specifies the number of bits to be transferred.

IDEST – An integer variable or array element that identifies the location to where the bits are to be transferred.

ISTRT2 – An integer expression that identifies the starting position within IDEST for the bits.

The low-order bit in either integer is zero (0). The last bit position in either integer must not exceed 31. (The length of an integer must not exceed 32 bits.)

Subroutine RBYTE (IBYTE, IVAL, IARY)

This routine sets the value of IVAL equal to the value contained in the element IBYTE of array IARY.

Functions and subroutines referenced: NONE

Subroutine READAL (ICOM, ICOLL, ICOLH, IWORDS, IDATA, IERR)

This is an auxiliary routine to DCODE that returns an alphanumeric value IDATA from the array ICOM based on the variables ICOLL, ICOLH, and IWORDS, which specify the location and size of the data string. An error code of 0 is returned in IERR if the operation is not successful.

Functions and subroutines referenced:

RBYTE SBYTE TEK_ADV

Subroutine RESTOR (IRECL,IRECH)

This routine reactivates the previously deleted records from IRECL to IRECH into the current data base. To reactivate a single record, let IRECL = IRECH.

Functions and subroutines referenced:

LBIT RANDIN RANDOU SETPRIM

Subroutine SBYTE (IBYTE, IVAL, IARY)

This routine sets the value of element IBYTE in the array IARY equal to the value of IVAL.

Functions and subroutines referenced:

LBIT

Subroutine SMVBITS (IVAL1, ISTART, ILEN, IVAL2 ITO)

This routine was designed to duplicate the VAX-specific subroutine that moves data bits from one location to another. The arguments are specified as follows.

IVAL1 – An integer variable or array element that contains the bits to be transferred.

ISTART – An integer expression that identifies the position of the first bit within IVAL1 to be transferred.

ILEN – An integer expression that specifies the number of bits to be transferred.

IVAL2 – An integer variable or array element that identifies the location to that the bits are to be transferred.

ITO – An integer expression that identifies the starting position within ISTART for the bits.

The low-order bit in either integer is zero (0). The last bit position in either integer must not exceed 31. (The length of an integer must not exceed 32 bits.)

Subroutine SYSCLK (NDATE, NTIME)

This routine returns the date in YMD (year, month, day) format in the variable NDATE. The present time in HMS (hour, minute, second) format is returned in the variable NTIME.

Functions and subroutines referenced:

IDATE TIME

Subroutine TYCON (IA, BI)

This routine converts the integer variable IA into a real number and the results are returned in the argument BI.

Functions and subroutines referenced: NONE

SECTION 4. SIMULATION CONTROL AND EXECUTION

The CASE/A solution system consists of several routines that work as a group to accomplish the simulation of an ECLS configuration. The functional grouping of the routines is as follows: (1) the execution control logic, (2) individual component routines, (3) component interface routines, and (4) specialized routines.

To obtain a system simulation, the system components and interconnects are first defined by the use of the graphical interface. After the system has been defined, the CASE/A solution system is invoked by the SOLVE command. The solution system performs a quasi-steady-state iterative solution at each time interval according to the control parameters specified in the CONTROL data file for the selected case.

The solution algorithm operates on an iterative component level basis to achieve convergence of the mass flows, temperatures, and pressures of each component stream to within the relaxation criteria specified in the CONTROL data. The user may disable the temperature or pressure convergence checks in the SOLVE command argument list if they are not of interest in the simulation. When the convergence checks are disabled, the properties are calculated for the component streams based on standard temperature and pressure. The mass flows are always checked for convergence within the component routines.

4.1 Execution Control Logic Description

When the SOLVE command is entered by the user, the main program command processor calls the routine named SOLVE. The SOLVE routine is then in control of program execution until the system simulation is complete or a fatal error condition is generated. The SOLVE routine manages the library of component routines that are called sequentially as the solution progresses in a specified "PCS." The SOLVE routine is arranged in distinct logic segments to accomplish the simulation. A flow chart for the solve routine is shown in figure 6.

4.1.1 Simulation Logic Segment Structure

The SOLVE routine first calls the routine SORTIEQ, that performs a sort on the list of components and connections data to order the components in descending priority. The assignment of subsystem components to cabins is then performed by calling the routine PTMOD. This feature allows heat transfer exchange to occur between the cabin environment and each component located in a subsystem schematic assigned to it. The PTMOD routine sets up pointers to various arrays that track the heat transfer between components and their assigned cabin environment. If a subsystem has not been assigned to a cabin, the thermal environment for that subsystem is set to 75 ˚F. Next, the CONTROL data base file is opened and the control parameters are read into memory. The control parameters include the simulation initial time, termination time, time step, output time interval, relaxation criteria, maximum number of solution iterations per time step, number of constituents to track, and the system damping factor. The values for these parameters are loaded from the data base file into the system variables described in table 1. The property and constituent arrays, summary arrays (routine SUMINIT), and hydraulic stream codes (routine PINIT) are then initialized. The component performance data initialization is then performed by the routine EQLOAD called for each component by SOLVE. The routine PSEUDO sets up the PCS and is called by the SOLVE routine just after the component data initialization has been performed. Any specialized initialization calculations required for the component solutions are then performed by the individual component routines at the direction of the SOLVE routine via the solution common block variable MFLAG. The value of MFLAG directs each component routine to perform one of four discrete functions listed below and described in chapter 9.

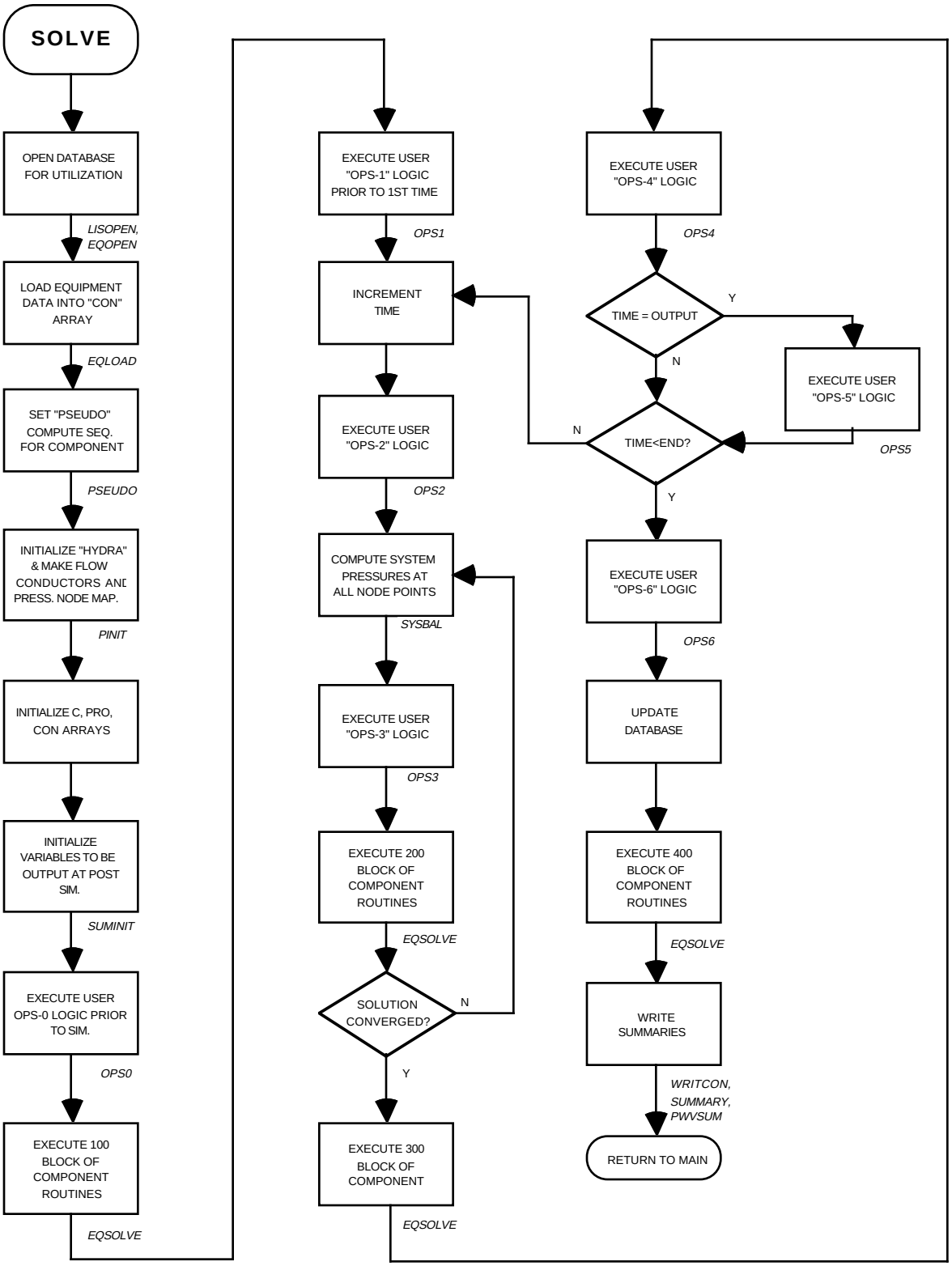


Figure 6. SOLVE routine flow chart.

Table 1. Simulation control variables.

NAME	DESCRIPTION
TIME	The current simulation time in hours.
STRT	The simulation starting time in hours (defaults to 0.0).
END	The simulation termination time in hours (defaults to 24.0).
STEP	The simulation time step interval in hours (defaults to 1.0).
RELX	The relaxation criteria to be met on a percent change basis from the previous iteration value. All component streams must meet this criteria for their mass flows, temperatures, and pressures.
IEQ	The current relative equipment number in the PCS. This variable is updated in the SOLVE routine.
LPCS	A variable that is used to step through the connection map data according to the PCS.
MFLAG	Indicates the simulation code segment to execute in the component routines.
RXCC	Tracks the maximum percent change of all component streams up to the current point in the PCS. This value must be below RELX for advancement to the next time step after a complete system iteration.
IPTFLG	Indicates whether the user wishes the system to perform relaxation checks on temperatures and/or pressures. The default value indicates checks are to be made for both in addition to the mass flow checks that are always made.
NEQ	The number of components in the case to be simulated.
NCON	The number of fluid streams in the case to be simulated.
NCOMP	The number of component types currently supported (54).
NTRACK	The number of constituents to be tracked (up to 50).
NSYSLOOP	The current number of system iterations performed during the present time step.
SYSDAMP	A damping factor used by a few specialized hydraulic routines.

MFLAG = 1	Component data initialization
MFLAG = 2	Iterative component level solution
MFLAG = 3	Post time step wrap-up
MFLAG = 4	Post simulation wrap-up
MFLAG = 5	Error occurred, wrap-up and return.

The iterative solution process begins after the component data initialization at the simulation start time and is continued until the termination time is reached. At each time interval, the SOLVE routine invokes the component routines sequentially according to the PCS until all of the components in the system have been processed. The routine EQSOLVE is responsible for calling the correct component routine based on the type of the component being solved for. After each complete system iteration, the convergence flag indicators are checked for compliance with the desired relaxation criteria. The iteration process is repeated until convergence is obtained for each and every component or until the maximum number of iterations is reached. The post time step wrap-up is performed after the iteration process is completed. The time is then incremented and the whole process is repeated until the simulation termination time is reached or a termination error condition occurs on the component level (indicated when a component routine sets MFLAG to 5). The progress of the simulation is indicated on the display terminal at the completion of each time step. A detailed report of the stream compositions and conditions for each component is written into a file CASENAME.LPP at each output time interval by the routine SSOUT. The simulation wrap-up is performed when the termination time has been reached. The final operations are performed by the routines SUMMARY and PWVSUM, that provide summaries of mass balance calculations and power usage, as well as weight and volume tabulations at the end of the LPP file. Execution control is then returned to the main command processor.

4.1.2 Pseudo-Compute Sequence

The PCS defines the order of the invocation of the component subroutines and may be modified by the user during the solution process. The PCS is initially set by the order in which the components are located in the system schematic in the case construction phase. The PCS is then resequenced in the SORTIEQ routine to order the components from highest to lowest priority. A priority code array inside the SORTIEQ routine sets the priority of each component type based on the characteristics of its operation. Components of equal priority are kept in the same relative order as when they were located in the system schematic. As the sorting process is executed, the component connection data are also updated to reflect the sorting of the relative equipment list. The user may change the PCS by the invocation of the SEQUENCE routine from the user operations blocks, which are discussed in section 4.1.3 and chapter 7.

After the relative equipment list has been sorted, the connection mapping is then performed by the PSEUDO routine. The PSEUDO routine translates the component connection data into an ordered map of the connections between the components based on the PCS. Pointers for the constituent array locations are established for all component streams. These pointers are used by all component routines to properly track the constituent mass flows of the system (see figure 7.3.2-1). A summary printout of the PCS and connection map is written to a file named CASENAME.CMP for use in debugging if required. An example printout is shown in figure 7.

4.1.3 User Operations Routines

The ability to incorporate user-defined logical and computational operations into the solution flow is provided by seven entry points in the SOLVE routine. The entry points permit the user to perform initialization and wrap-up functions on a per-simulation basis, to perform time varying operations on a per-time-step basis, to perform operations within the solution of each time step, and to write out plot data. The relationship of the seven user-defined operations blocks to the overall solution flow is illustrated in figure 6. The seven blocks are discussed in chapter 7.

```
**********************************************************************
                    CONNECTION MAP FOR CASE ATC_7

**********************************************************************
    NUMBER OF COMPONENTS    4     NUMBER OF CONNECTIONS   5
    LENGTH OF PSEUDO    6


                          COMPONENT LISTING

REL                      CENTROID   CONNECTION OF EACH COMPONENT STREAM
IEQ  SUBSYS  ID   NAME    X    Y     1    2    3    4    5    6    7    8
--- -------- -- -------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  1 ATC      27 P-1       301  426    5    1    0    0    0    0    0    0
  2 ATC      32 LOAD-1    651  206    1    4   14    0    0    0    0    0
  3 ATC      33 PAYLOAD   650  554    4    5    0    0    0    0    0    0
  4 ATC      50 C-1       607  420    0    0    0    0    0    0    0    0


                          CONNECTION LISTING

      REL              REL
CONN  IEQ STREAM  TO   IEQ STREAM  NAME OF COMPONENTS
---- ---- ------     ---- ------  ------------------
   1    1    2   ==>    2    1    P-1     : LOAD-1
   2    2   -2   ==>    4    2    LOAD-1  : C-1
   3    4    1   ==>    1   99    C-1     : P-1
   4    2    2   ==>    3    1    LOAD-1  : PAYLOAD
   5    3    2   ==>    1    1    PAYLOAD : P-1


                       PSEUDO COMPUTE SEQUENCE

SEQUENCE    CONN     REL             REL
 NUMBER    (NCPT)   IEQ  STREAM  TO IEQ  STREAM
--------  --------  ---- ------     ---- ------
     1         5       1     1  ==>    3     2
     2         1       1     2  ==>    2     1
     3         1       2     1  ==>    1     2
     4         4       2     2  ==>    3     1
     5         4       3     1  ==>    2     2
     6         5       3     2  ==>    1     1
```

Figure 7. Example connection map printout.

4.2 Solution System Library Routines

CASEAMAIN

The main program serves as the command processor/user interface for the CASE/A system. Commands and their arguments are typed after a command line prompt by the user and processed by CASEAMAIN. Upon completion of the command control is returned to CASEAMAIN and a new prompt issued. The variable NCODE, a two-dimensional four-byte integer, contains a “list” of possible commands. The first four characters are referenced by the first index of NCODE and the second four characters are referenced by the second index. Thus, a command can be a maximum of eight characters long. Notice also that the command is text but stored as an integer. The command and its arguments are read from the input line with a read statement into the character variable ICOM. This character string is passed to DCODE to determine the first “word” on the command line and returned in the variable ITEXT. The NCODE list of commands is searched for a match to ITEXT and the variable I is set equal to the relative position of the command in this list. Control is passed to a statement label using a computed

GOTO on the value of I. For instance a value of I = 1 would transfer control to statement label 201, I=2 goes to 202, etc. up to I=134. Upon completion, control is returned to statement label 20 where a new prompt is issued. This process is repeated until the user issues the “EXIT” command or presses CONTROL-Y. Notice also that commands are recognized only if they are type exactly as stored in the NCODE array. They are, however, translated from lower to upper case using the TOUPPER routine.

Functions and subroutines referenced:

ALARM	ASSIGN	ARCHIVE	AUTOPLOT
AVRAGE	BCK_GD_CLR	BLDREC	CALC
CASEREAD	CDCODE	CEDIT	CHANGE
CHKREC	CILLCHAR	CLCALC	CLOADCASE
CLONE	CLS	CONECT	
COPYALL	CNEWCASE	CREATE_KEYBD	CTOUPPERC
CUTALL	DBOPEN	DCALC	DCLFOR
DCODE	DELCAS	DELCN	DELEQ
DELETE	DELLAB	DELNOTE	DELREC
DIR	DITTO	DTABLE	EDITOR
EDT	ELIMINAT	ENTRY	EQOPEN
EXPORT	FAST	FDUMP	FILE
FILEREC	FIND	FIT	FLAG
FRAME1	FULL	GRALPH	GRCHRZ
GRCOPY	GRERAS	GRINIT	GRLBCT
GRMOVE	HELP	IMPORT	INPUT
INTERSEC	JOIN	KBIT	LABL LBIT
LIB$SYS_TRNLOG	LIST	LISTAB	LOAD
LOCATE	MASKER	MERGE	MERGE_IN
MERGE_OUT	MODBAK	MOVALL	MOVEIT
MOVLAB	MOVNOTE	NEWCASE	NOTE
OPENDB_X	PLTOVR	POINT	POLYSIM
PRELIM	PREPORT	PREPRO	PRIMDB
PRNTSS	RAMDISK	RANDIN	RANDOU
RANGE	RANK	RBYTE	
READCOMM	REDRAW	RENAME	RENEQ
REPORT	RESTOR	RETRIEVE	RNSS
ROTATE	SAMFST	SAMPLE	SAVE
SAVEAS	SCREDT	SECURE	SELECT
SETPRIM	SIGMA		
SMG$CREATE_COMPOSED_LINE		SMG$CREATE_KEY_TABLE	
SMG$CREATE_VIRTUAL _KEYBOARD			SOLVE
STATS	SUBSYS	SUBTTL	SYSCLK
TABLE	TALLY	TARGET	TDPLOT
TECREPORT	TEK_ADV	TOGGLE_SCREEN	TYPE
UNASSIGN	UNION	UPDPLT	WRITCON

Subroutine DPCS(IEQA,ISTA,IEQB,ISTB,NCPT)

Used to unload the IPCS array PCS.

Functions and subroutines referenced:

SMVBITS

Subroutine EQSOLVE

EQSOLVE calls the COMPONENT routines using a computed GOTO based on the value of ITYPE. ITYPE is determined for each component from the EQL array. This routine is called from SOLVE inside a DO LOOP, which executes once for each component in the system.

Functions and subroutines referenced:

ADSORPTN	AFSPE	BMR	BOSCH
CABIN	CAP	CFR	CHX
CNHX	CNTRLLR	CO2LIQ	CP
CREW	DEFLOW	DEHUM	EDC EVAP
FCELL	FILTER	FINDC	FOODPROC
H2OSEP	HATCH	HEATER	HX
HYDISS	IONEXCH	LIOH	MODULE
MOLSIEV	MSPLT	NODE	O2N2 OPS7
PIPE	PLANT	POINTCON	PREWAST
PUMP	RACK	RAD	RO
SABAT	SAWD	SFWE	SINK
SOURCE	SUM	TANK	TBUS
TIMESC	TIMER	VALVE	VCD
WASH	WQM		

Subroutine PSEUDO

The primary interface routine between the graphic and solution system.

Functions and subroutines referenced:

CMPOPEN	POINTCON	SORTIEQ	SMVBITS
TEK_ADV			

Subroutine SEQUENCE(NAME)

This routine can be invoked by the user in OPS logic to resequence the solution order.

Functions and subroutines referenced:

TEK_ADV

Subroutine SOLVE

The primary driver for the solution system.

Functions and subroutines referenced:

CASEREAD	CASEWRIT	CNTRLLR	DUPLICATE
EADSAVE	EQLOAD	EQOPEN	EQSOLVE
FINDC	FOR$CLOSE	FRAME1	GRERAS
HEADER	LISOPEN	OPS0	OPS1 OPS2
OPS3	OPS4	OPS5	OPS6 PINIT
PLTDATA	PLTFILE	PIONTCON	PSEUDO
PWVSUM	RANDIN	RANDOU	SORTIEQ
SSOUT	SUMINIT	SUMMARY	SYSBAL
TEK_ADV	TIMER	WRITCON	

Subroutine SORTIEQ

Used to sort the PCS to put the active components first.

Functions and subroutines referenced:

PTMOD SAVE

SECTION 5. MODEL OUTPUT MANAGEMENT

The following sections discuss the routines involved in the forms of output supported by CASE/A.

5.1 Schematic Output

There are several routines available to the user regarding schematic output. These include screen refreshing, generating hard copies of subsystem layouts, and locating stream data on the schematic. The commands are discussed in section 5.1 of the User's Manual and the subroutines that support them are detailed in section 2 of this manual.

5.2 Component Data Base Output

It is sometimes useful to obtain a hard copy of the data base contents for components. CASE/A provides the PRINTSS command, which prints a copy of the data base contents for each component of a specified subsystem. The routine to support this function is discussed below.

Subroutine PRNTSS

This routine is used to print the component data for a subsystem.

Functions and subroutines referenced:

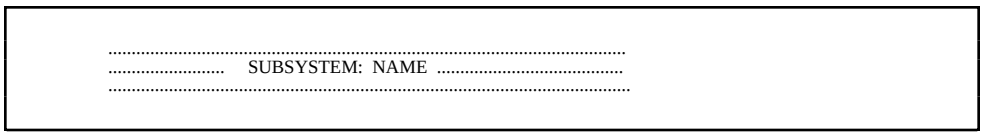
CDCODE	CILLCHAR	CTOUPPERC	EQOPEN	ITMOUT
RANDIN	TEK_ADV			

5.3 Simulation Summary

The solution routine (SOLVE) generates the system summary file CASENAME.LPP. This file is generated by calling HEADER and SSOUT at each output interval. The routines WRITCON, SUMMARY, and PWVSUM are called after the simulation has ended. Each of these routines is discussed below. See chapter 5.3 of the User's Manual for an example .LPP file.

Subroutine HEADER

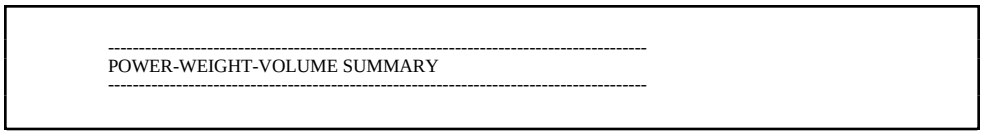
This routine writes the ".LPP" file header. This is done at each output interval. This section consists of the values of TIME, STEP, RELXX, RELCC, LOOPS. and the subsystem name in the form:



Functions and subroutines referenced: NONE

Subroutine PWVSUM

This routine develops the power and weight summaries during simulation wrap-up. This summary is at the end of the .LPP file and begins with



It follows with total P, W, V for each subsystem and the total for the case.

Functions and subroutines referenced:

SMVBITS

Subroutine SSOUT(NA)

SSOUT is used to output the individual component data to the .LPP file. NA is the subsystem name. This section of the LPP file comes directly after the header and prints the case name, component name, type, IEQ number, and subsystem name, along with the values from the C and PRO array for each of the component streams. This section begins with a row of equal signs for each component.

Functions and subroutines referenced:

SMVBITS PROPS

Subroutine SUMMARY

Performs the summary logic for a simulation wrap-up and print out. This routine prints summary data for components that accumulate mass, for example tanks and cabins. The list of data to be summarized is contained in the variable descriptions located in appendix A of this manual. This section comes prior to the PWV summary at the end of the simulation. It is easily found in the file as it is delimited with a row of #'s.

Functions and subroutines referenced:

SMVBITS

Subroutine WRITCON

This routine updates the CON array with output and benchmark values for each component. It then writes the values to the .LPP file. This appears below the last SSOUT entry and before the first SUMMARY entry. This section is delimited with a row of +'s.

Functions and subroutines referenced.

EQOPEN ISTAT RANDIN RANDOU TEK_ADV

5.4 Integrated Plot Utility

The integrated plot utility (IPU) provides the capability to save specified data for analysis and plotting. Details for creating "PLOTSETS" and using IPU commands are given in section 5.4 of the User's Manual. To create the data for the IPU, the SOLVE routine calls two subroutines. It calls PLTFILE before loading the component data and just after loading the PLOT data base into an array (IA). After each OPS call, SOLVE calls PLTDATA if the plotset is active for that OPS logic block (see section 4.3 of the User's Manual). PLTDATA writes out, in binary form, the data requested by the user in the PLOTSET. To use the IPU to generate plots on the screen or printer copies of the plots, the IPU

command is given at the CASE/A prompt. This action sets the logical variable TDMS_FLAG to true. When TDMS_FLAG is true, the IPU commands discussed in section 5.4 of the User's Manual become active. This also causes other commands, such as "SOLVE" to be disabled. These IPU commands are documented in appendix B.2 of the User's Manual. Following is a brief description of PLTFILE and PLTDATA. Variables used by the IPU are tabulated in appendix A of this manual.

Subroutine PLTDATA(NUM,PLD_ERR)

This routine executes a loop from 1 to KOUNT(NUM) (Number of items in the PLOTSET) and loads the DATA array with the data specified in the variable IPLT. When the data are extracted, the record is written to the .DAT file.

Functions and subroutines referenced:

FINDC GETPP OPENDB_X RANDOU RESTOR SETPRIM

Subroutine PLTFILE(IA,PLF_ERR)

This routine initializes the PLOTSET arrays, pointers, and creates the .DAT, .STT, .SCR and .DDF files. PLF_ERR is set to 1 if an error occurs.

Functions and subroutines referenced:

CDCODE CTOLOWERC GETI OPENDB_X RANDIN SETPRIM TEK_ADV

5.5 Data Output Options

When using OPS logic, the need may arise to store user-defined variables for later analysis. These variables, however, may not be accessible to the IPU through any of the system data arrays (CON, C, PRO, etc.). In such cases, the user has two primary methods to save this data for use with TDMS:

(1) Output to the USERCON array and

(2) Custom user output (e.g. with FORTRAN write statements).

5.5.1 Output to the "USERCON" Array

The USERCON array is the preferred method of making user-defined variables available to the IPU. The USERCON array is a one-dimensional array of 100 elements. This array is declared in a COMMON BLOCK in all of the appropriate "INCLUDE" files (see appendix) and is therefore automatically available to the user in all OPS subroutines. Additionally, any other user-provided subroutine may access this array by adding the following statement to its variable declaration code:

```
INCLUDE 'CASEA$CODE:COMPCOM.INC'
```

Locations 1 to 60 of USERCON are available in the USERCON data base (accessed while running CASE/A by giving the command "ED;USERCON"). Values may be entered through the CASE/A editor and used directly in OPS logic. Similarly, output values may be stored in the USERCON array and viewed in the data base after a simulation is complete. For real-time data storage, the desired parameters should be stored in the USERCON array and accessed by the IPU (CODE=USER). See section 4.3 of the User's Manual for a description of this process. Locations 61 to 100 are accessible by the

IPU but are not displayed as part of the USERCON data base when viewed in the full-screen mode (due to screen size limitations only).

As an example, suppose the temperature of the fluid stream at the outlet of a PIPE, named "PIPENAME", in a given model is to be tracked in Celcius rather than Fahrenheit. The conversion should be performed in OPS4 (post time step). The code in figure 8 converts the temperature and stores the value in location 3 of USERCON.

```
Subroutine OPS4
...
CALL GETT('PIPENAME',2,TEMPF)
TEMPC = 5.0/9.0 * (TEMPF - 32.0)
USERCON(3) = TEMPC
...
RETURN
END
```

Figure 8. Variable output to the USERCON array.

The plotset definition screen (ED;PLOT) entries necessary to output this value to a file is shown in figure 9.

```
                        PLOTSET DEFINITION

CODE  COMPONENT NAME    LOCATION    STREAM#          TITLE
----  --------------    --------    -------    ------------------------
USER                      0003         0       Pipe Outlet Temp,  C
```

Figure 9. Example PLOTSET definition for the USERCON array.

5.5.2 Custom User Output

If a particular output situation does not lend itself to the IPU, the user has two options for writing output directly to data files. The data may be written to an ASCII text file or written in binary form so that a data base is created that can be accessed with TDMS as described below.

5.5.2.1 Writing Data to an ASCII Text File

The simplest method of saving user-defined data is through standard FORTRAN formatted WRITE statements. Using this method, the data could be written to the file in such a way as to be recognizable to some commercial data analysis package. If no postprocessing is desired, the output could simply be formatted for direct incorporation into a presentation or report. For the temperature conversion example above, the process might be as simple as shown in figure 10.

```
      Subroutine OPS0
      ...
      OPEN (UNIT=60,FILE='MYOUTPU.DAT',STATUS='NEW')
      ...
      RETURN
      END

      Subroutine OPS4
      ...
      CALL GETT('PIPENAME',2,TEMPF)
      TEMPC = 5.0/9.0 * (TEMPF - 32.0)
      WRITE(60,100) TEMPC
100   FORMAT (1X, 'EXIT TEMPERATURE OF PIPENAME = ', F7.2, ' F')
      ...
      RETURN
      END
```

Figure 10. Writing custom data to a text file.

5.5.2.2 Creating a Custom Data Base for Storing Output Data

There are two steps in creating a custom data base. The first step is to write the desired parameters to a binary file. The second step is to create a data definition file (.DDF) that specifies how the data are stored. The following example illustrates these procedures.

Consider a PUMP-STORE connection that is part of some larger model. The PUMP (named P1) is pumping a fluid composed of oxygen, nitrogen, carbon dioxide, and water vapor (of unknown concentrations) and directing it into the STORE (named S1). For this example, the components upstream of the pump and downstream of the storage tank are unimportant. Suppose the user wishes to determine the flow rate and stream composition at the pump outlet, and the storage tank temperature and pressure as a function of time. These parameters, the associated CASE/A arrays, and the locally defined user variables are shown in figure 11.

Parameter	CASE/A	Location	Stream	Variable
time	TIME	-	-	TIME
flow rate	C	1	2	MDOT
O2 mass frac	C	2	2	FO2
N2 mass frac	C	3	2	FN2
CO2 mass frac	C	4	2	FCO2
H2O mass frac	C	8	2	FH2O
tank temp	PRO	2	1	S1T
tank press	PRO	1	1	S1P

Figure 11. Example variable output using binary files.

Recall that the components are named P1 (PUMP) and S1 (STORE) and that the data are to be saved as a function of time. Thus, the code in figure 12 would be placed in OPS4 of the users supplied operations logic. See section 7.0 of the user's manual for a description of the utility routines.

```
      Subroutine OPS4
      ...
      CALL GETC('P1       ',2,1,MDOT)
      CALL GETC('P1       ',2,2,FO2)
      CALL GETC('P1       ',2,3,FN2)
      CALL GETC('P1       ',2,4,FCO2)
      CALL GETC('P1       ',2,8,FH2O)
      CALL GETC('P1       ',1,S1T)
      CALL GETC('P1       ',1,S1P)
      ...
      RETURN
      END
```

Figure 12. Example code to get values for pump "P1."

Now that the data are available in local variables, it can be written to a binary file as shown below.

WRITE (99) TIME, MDOT, FO2, FN2, FCO2, FH2O, S1T, S1P

This WRITE statement outputs the data to a binary file named FOR099.DAT. Although the use of logical unit number 99 is arbitrary, it is recommended that only values above 60 be used to prevent conflict with CASE/A file access routines.

The second step of creating the data base is the development of the data definition file. Figure 13 presents the .DDF, named EXAMPLE.DDF, for the problem at hand.

```
010         010         009         15000       010   0     0
003         001         001         TIME, HRS
003         001         002         PUMP OUTLET FLOW RATE LB/HR
003         001         003         PUMP OUTLET O2 FRACT
003         001         004         PUMP OUTLET N2 FRACT
003         001         005         PUMP OUTLET CO2 FRACT
003         001         006         PUMP OUTLET H2O FRACT
003         001         007         TANK TEMPERATURE, F
001         001         008         TANK PRESSURE, PSIA
003         001         009         MODIFICATION DATE
002         001         010         SECURITY
```

Figure 13. Example data definition file for custom output.

After the simulation is completed, the data base may be activated by entering the IPU, or by running TDMS as a standalone package. TDMS will ask the user to enter a data base name. This is the same as the .DDF name, i.e., EXAMPLE (with no extension). Next, the data must be pulled from the FOR099.DAT file and stored in the data base. To do this, use the SELECT command (see the on-line help for assistance).

```
SELECT;99;8
```

The first parameter is the FORTRAN logical unit number, and the second number reflects the number of items in the WRITE statement. Note that the .DDF may be longer than necessary (i.e., support more items than actually contained in the WRITE statement) to allow room for expansion. If the .DDF is shorter, however, the data will wrap from record to record and the data base will be corrupted.

Once the SELECT command has been executed, the data base is complete. The next time the data base is accessed, the SELECT function is not needed. If the simulation is run again, however, the SELECT process must be repeated to load the new data into the data base.

5.6 Schematic Connection and Hydraulic Maps

The CASE/A solution routine automatically generates two output files containing schematic connection and hydraulic flow conductor information to assist the user in debugging new or existing cases. The .CMP and .FMP files for the case schematic shown in figure 14 will be discussed in the following two sections.

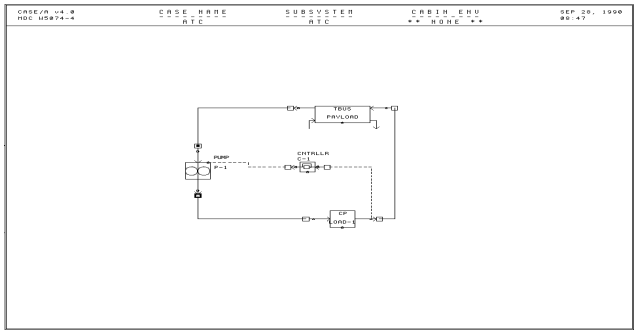


Figure 14. Example subsystem schematic.

5.6.1 The .CMP File

The schematic connection map is created in the user's default directory and named CASENAME.CMP. An example .CMP file is shown in figure 15. The first line identifies the case name. The next two lines list the total number of components and connections located in the case along with the length of the PCS, which is discussed here and in section 4.1.2. The next section in the .CMP file lists all of the components in the case by relative equipment number (IEQ) as determined in the SORTIEQ routine (see section 4.1.2). Component listing information consists of the relative IEQ number, subsystem name, component type (ITYPE), component name, (x,y) coordinates (in pixels) of the component icon body centroid relative to the lower left hand corner of the terminal screen, and the connection numbers of the component streams (maximum of 8). Notice that the inlet connection (stream 1) to the PUMP named P-1 has the same connection number (1) as the outlet connection (stream 2) of the TBUS named PAYLOAD.

```
**********************************************************************

                    CONNECTION MAP FOR CASE ATC_7

**********************************************************************
    NUMBER OF COMPONENTS    4     NUMBER OF CONNECTIONS   5
    LENGTH OF PSEUDO    6


                         COMPONENT LISTING

REL                        CENTROID   CONNECTION OF EACH COMPONENT STREAM
IEQ  SUBSYS  ID   NAME      X    Y     1    2    3    4    5    6    7    8
--- -------- -- -------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  1 ATC      27 P-1       301  426    5    1    0    0    0    0    0    0
  2 ATC      32 LOAD-1    651  206    1    4   14    0    0    0    0    0
  3 ATC      33 PAYLOAD   650  554    4    5    0    0    0    0    0    0
  4 ATC      50 C-1       607  420    0    0    0    0    0    0    0    0


                         CONNECTION LISTING

      REL             REL
CONN  IEQ STREAM  TO   IEQ STREAM  NAME OF COMPONENTS
---- ---- ------     ---- ------  ------------------
   1    1    2   ==>    2    1    P-1     : LOAD-1
   2    2   -2   ==>    4    2    LOAD-1  : C-1
   3    4    1   ==>    1   99    C-1     : P-1
   4    2    2   ==>    3    1    LOAD-1  : PAYLOAD
   5    3    2   ==>    1    1    PAYLOAD : P-1


                      PSEUDO COMPUTE SEQUENCE

SEQUENCE    CONN     REL             REL
 NUMBER    (NCPT)   IEQ  STREAM  TO IEQ  STREAM
--------  --------  ---- ------     ---- ------
     1         5      1     1  ==>    3     2
     2         1      1     2  ==>    2     1
     3         1      2     1  ==>    1     2
     4         4      2     2  ==>    3     1
     5         4      3     1  ==>    2     2
     6         5      3     2  ==>    1     1
```

Figure 15. Example connection map (.CMP file).

The connection listing of the .CMP file simply shows each connection located in the case. In the first column is the absolute connection number followed by the relative IEQ numbers and stream numbers of the two components that are connected. The names of the components being connected are given in the last column. Notice for the controller (CNTRLLR) that a “data” connection to a component stream is assigned the negative of the stream number (-2 in this case) and a “data” connection to a component body (controller to pump) results in a stream number of 99. This results in CASE/A not tracking these “data” connections in the PCS. The last data block in the .CMP file is the PCS. The first column is the PCS number followed by the connection number (NCPT = the number of the connection pointer) and the component streams that are connected. The PCS is determined by the equipment sort performed by SORTIEQ and by the number of connections made to that component. For example, a CABIN component located in a case would receive the highest ranking in SORTIEQ. Then all of the connections made to that CABIN would appear next in the PCS in ascending order of the relative IEQ of the connecting component. The PCS, then, is a list, in order of their relative equipment number, of all the components connected to streams of a component. In the example, for instance, the first component in the PCS is

IEQ 1. It has two entries in the PCS, sequence numbers 1 and 2. This component, a pump with priority 2, has two streams. Stream 1, the inlet, is connected to IEQ 3, the exit of the TBUS (stream 2). The exit stream (2) of the pump is connected to the inlet stream (1) of the cold plate, whose IEQ is 2. Generally, the PCS has a length equal to twice the number of connections. Thus, it is seen that the PCS is an ordered list of components that shows how the streams of a component are connected to other components. The array NPCS stores the component IEQ numbers in the order they appear in the PCS. This is the SOLUTION ORDER and can be changed with the SEQUENCE subroutine. This array is used by SOLVE to determine the next IEQ in the solution process. The "SEQUENCE NUMBER" of the PCS should not be confused with the SOLUTION ORDER contained in the NPCS array. Note that controller (CNTRLLR) and time (TIMER) components do not appear in the PCS. For a BUBBLE connection, the connection number on the "B" side of the BUBBLE is used in the PCS and the BUBBLE is removed. In summary, the PCS is simply an orderly listing of all connections that each component (excluding CNTRLLR, TIMER and between BUBBLE's) "sees" both upstream and downstream, sorted by relative IEQ number.

5.6.2 The .FMP File

The flow conductor map is created in the user's directory and named CASENAME.FMP. An example .FMP file is shown in figure 16. The first line identifies the case name. The next seven lines give the total number of components, pressure nodes, flow conductors, flow subnetworks, controllers, timers, and bubbles located in that particular case. Pressure nodes correspond to the fluid stream hit boxes of each component in the case. CNTRLLR and TIMER hit boxes, of course, are not included in the hydraulic solution since they represent the flow of data. A BUBBLE component simply represents a flow connection across subsystem screen boundaries and, thus, is not included as a pressure node. The pressure node information given in the .FMP file includes the system node number, component number, component relative equipment number (IEQ), subsystem name, component type (both ITYPE and the alphanumeric name), component name, component stream number, and connection number (NCPT) attached to that stream. The flow conductor information list contains the system conductor number, the two system nodes attached to that conductor, and the component names and stream numbers that correspond to those two nodes. The entire hydraulic layout of the case (nodes and connectors) are subdivided into separate subnetworks that are terminated by boundary pressure nodes as described in section 8.1.1 of this manual and also in section 8.2.1 of the User's Manual. The flow subnetwork information in the .FMP file is broken into separate sections for each subnetwork. In this example, there is only one network. The first line contains the subnetwork number and the number of nodes and conductors that it contains. A mapping is then provided that shows how the network specific node and conductor numbers correspond to the actual system node and conductor numbers. It is highly advisable to have the case schematic(s) close at hand when following the information provided in the .FMP file. The user is encouraged to mark the node and conductor numbers on the schematic(s) to help visualize the hydraulic subnetworks that CASE/A is attempting to solve. The .FMP file is an extremely valuable debugging tool to aid in solving pressure convergence problems.

```
************************************************************************

                        FLOW MAP FOR CASE ATC_7

************************************************************************
    NUMBER OF EQUIPMENT ITEMS:       4
    NUMBER OF PRESSURE NODES:        6
    NUMBER OF FLOW CONDUCTORS:       5
    NUMBER OF FLOW SUBNETWORKS:      1
    NUMBER OF CNTRLLR COMPONENTS:    1
    NUMBER OF TIMER COMPONENTS:      0
    NUMBER OF BUBBLE COMPONENTS:     0


    *** NOTE: CNTRLLRs, TIMERs, AND BUBBLEs DO NOT  ***
    *** APPEAR IN THE NODE INFORMATION LIST BELOW   ***


                      PRESSURE NODE INFORMATION

SYSTEM REL                                              CONN
 NODE  IEQ  SUBSYS  COMPONENT TYPE   NAME   STREAM (NCPT)
------ --- -------- -------------- -------- ------ ------
    1    1 ATC       27  PUMP      P-1          1      5
    2    1 ATC       27  PUMP      P-1          2      1
    3    2 ATC       32  CP        LOAD-1       1      1
    4    2 ATC       32  CP        LOAD-1       2      4
    5    3 ATC       33  TBUS      PAYLOAD      1      4
    6    3 ATC       33  TBUS      PAYLOAD      2      5


                      FLOW CONDUCTOR INFORMATION

 SYSTEM   SYSTEM    SYSTEM
CONDUCTOR NODE A TO NODE B  COMPONENT A STREAM TO COMPONENT B STREAM
--------- ------    ------  ----------- ------    ----------- ------
     1        1  ==>    6    P-1           1   ==> PAYLOAD       2
     2        2  ==>    3    P-1           2   ==> LOAD-1        1
     3        3  ==>    4    LOAD-1        1   ==> LOAD-1        2
     4        4  ==>    5    LOAD-1        2   ==> PAYLOAD       1
     5        5  ==>    6    PAYLOAD       1   ==> PAYLOAD       2


                     FLOW SUBNETWORK INFORMATION

============================================================

NETWORK NUMBER    1 CONTAINS    6 NODES AND    5 CONDUCTORS

  NETWORK    CORRESPONDS    SYSTEM
NODE NUMBER      TO       NODE NUMBER
 -----------  -----------  -----------
      1          ==>           1
      2          ==>           6
      3          ==>           5
      4          ==>           4
      5          ==>           3
      6          ==>           2

  NETWORK    CORRESPONDS    SYSTEM
 CONDUCTOR      TO        CONDUCTOR
 -----------  -----------  -----------
      1          ==>           1
      2          ==>           5
      3          ==>           4
      4          ==>           3
      5          ==>           2
```

Figure 16. Example hydraulic flow map (.FMP file).

SECTION 6. UTILITY COMMANDS AND MISCELLANEOUS SYSTEM ROUTINES

This section describes the commands available within the CASE/A environment that are designed to make system use more efficient, and those utility routines that are used by other subroutines. These commands are not always required for normal simulation execution, but perform useful functions such as providing an interface to VMS or examining various data.

6.1 Terminal Settings

The CASE/A modeling package is designed to operate on terminals capable of emulating Tektronix 4014 graphics terminals and/or DEC VT100 text terminals. The preferred configuration would be an advanced graphics terminal capable of emulating both (such as the composite terminal). The default terminal type is a GraphOn composite terminal. The terminal type may be changed by the user to match a particular system. The user may examine the status of a variety of terminal-related settings using the FLAGS command. Complete descriptions of these commands are provided below.

6.1.1 Terminal Setting Command

The TERM command defines the type of terminal used in a given CASE/A work session to permit the proper interpretation of editing and output commands. The following is a summary of the terminal types currently supported.

TERM 1 – Tektronix 4014
TERM 2 – VT100
TERM 3 – DUAL VT100/4014

When the TERM command is issued, the main program decodes the command line and sets the terminal characteristics according to the argument as listed above at statement label 246. The variable ITMNL, which is used by other routines as well, is set equal to the argument of the command (e.g., TERM;2 gives a value of 2 to ITMNL). The screen is initialized and cleared based on the value of ITMNL. There is no “TERM” subroutine such as is the usual case for most commands. The following routines are called, however, by CASEA_MAIN to set the terminal type:

CLS	CDCODE	LBIT	FRAME1
TEK_ADV	TOGGLE_SCREEN		

6.2 Miscellaneous Commands

The following describes some useful miscellaneous commands.

6.2.1 Flags

This command invokes the subroutine FLAG to display a list of 15 system flags. The list displays the status of the flag—either ON or OFF. The following flags are displayed:

LOGO,	INVERSE FLAG,	COMMAND ECHO,
AUTO HARD COPY,	PLOT TIME,	DUAL Y AXES,
VECTOR GRAPHICS,	X AXIS LOG SCALE,	Y AXIS LOG SCALE
OVER-WRITE PROTECT	BOLD PLOT	PAUSE FOR COPY
NASA LOGO	COLOR FLAG	
TASK COMPLETION ESTIMATE		PLOT LEGEND

These flags are discussed in the User's Manual in appendix B.1 and B.2.

Subroutine FLAG

The FLAG subroutine displays a list of system flags.

Functions and subroutines referenced: NONE

6.2.2 On-Line Help Information

The help routine is called by CASEA_MAIN and gives an explanation of all CASE/A commands including IPU unique commands. Included under each HELP topic is an explanation of the command's FUNCTION, the command syntax (FORMAT), and any helpful notes. Help is invoked by the command

CASEA> HELP;topic

If the topic is left blank, HELP will provide a list of available topics.

HELP opens an ASCII text file with the name TOPIC.HLP. These files are located in the CASEA$CODE directory.

Subroutine HELP

The HELP subroutine displays the contents of the file TOPIC.HLP.

Functions and subroutines referenced:

CDCODE	CLOSE	CLS	CTOLOWERC
CURSOR	FRAME1	GRALPH	GRMOVE
INDEX	OPEN	TEK_ADV	

6.2.3 VAX/VMS Commands

The user may temporarily exit the CASE/A environment and enter the VAX operating system by executing the DCL command. The subroutine DCLFOR, called by CASEA_MAIN, prompts for a DCL command and passes it, unchanged, to LIB$SPAWN. The command is executed and, upon completion, control is returned to DCLFOR. The user remains in DCLFOR until the command "LOGOUT" or "LO" is given.

The user should have a subprocess quota limit that allows a process to be spawned. If this quota is exceeded, the command will not be executed. Normal VMS diagnostic messages are displayed for errors, etc., however, no message is given if the process cannot spawn.

It should also be noted that commands that affect only a process, such as SET DEFAULT, will serve no purpose as they are only valid for the current process that is terminated upon completion.

Subroutine DCLFOR

This routine establishes the call to the VAX library function to allow execution of DCL commands.

Functions and subroutines referenced:

FRAME1 TEK_ADV

6.2.4 Temporary Exit to VAX Editor

The user may temporarily exit the CASE/A environment and directly enter the VAX/VMS text editor EDT. This is done by invoking the EDT editor from the subroutine EDT with the function EDT$EDIT(filename). This command allows the user to examine the output file CASENAME.LPP or modify the operations file CASENAME.FOR. Any other text file that is available to the user may be edited by supplying the appropriate path and filename. Since this is an invocation of the EDT editor, all normal EDT functions exist. When the user enters “EXIT” or “QUIT” after a CTRL-Z in the editor, control is returned to CASEA_MAIN.

Subroutine EDT

This command routine allows the user to edit any file on VAX mass storage. The file name is specified in the command argument list.

Functions and subroutines referenced:

CDCODE EDT$EDIT FRAME1 INDEX
TEK_ADV

6.3 Simulation Control Commands

There are several commands that assist in model development and simulation control. These commands involve individual components such as CABINS and entire cases.

6.3.1 Subsystem Heat Load Assignment to CABINS

The CABIN component is used to simulate a cabin, compartment, or otherwise isolated environment by tracking the respirable atmospheric compositions, mass additions/losses, and heat transfer between equipment, other cabins, and the external orbital environment. For the CABIN to recognize the equipment contained in it, the equipment must be assigned to the CABIN using the ASSIGN command, as described below. For a complete description of the interactions between cabins and the associated equipment, see section 10.0 of the User’s Manual.

6.3.1.1 ASSIGN Command

This command assigns a subsystem to a cabin. This provides the capability to address multiple cabins with different configurations and/or environments. The ASSIGN command must be issued before executing SOLVE. If the user fails to make a cabin assignment, the cabin air, structure, and mean radiant environment will default to 75 ˚F from a component standpoint. The user should zero out the environmental conductances for the components that are assumed to operate adiabatically. If the interaction with the environment is disabled, then the ASSIGN command is not required.

Subroutine ASSIGN

This subroutine loads the NSSCAB array with the appropriate subsystem name, cabin name, and relative cabin number of cabins that have at least one subsystem assigned to them. This information is used to determine which subsystems will interact thermally with a given cabin. NUMSS is the number of subsystems that have been assigned to a cabin. NUMCAB is number of cabins that have at least one subsystem assigned to them.

Functions and subroutines referenced:

CDCODE CILLCHAR CTOUPPER SAVE TEK_ADV

6.3.1.2 UNASSIGN Command

This command unassigns a subsystem from a cabin. It is functionally opposite of the ASSIGN command.

Subroutine UNASSIGN

This subroutine is functionally opposite of subroutine ASSIGN.

Functions and subroutines referenced:

CDCODE CILLCHAR CTOUPPER SAVE TEK_ADV

6.3.2 MERGE Operation

The MERGE command is particularly useful when developing large and complicated models. Complete subsystems are constructed and verified as small, manageable cases. The individual subsystems are then merged into the larger model. The smaller case may reside in the same directory, a different directory, or in a directory on a different VAX host than where the larger destination model exists. This command allows the user to include a complete subsystem (or the entire model) from a different case in the current case. Complete subsystems can be built and verified as separate cases and then the subsystem merged into the larger model. Development time is reduced since the models are developed as smaller, more manageable problems and then assembled into a single model. The routine will check for duplicate component names and solicit a new name from the user if necessary. If the user does not want to change the name, a carriage return is entered and execution continues.

The MERGE routine queries the user if the source case is from an older version. Then the routine prompts the user to enter the location of the source CASENAME.MOD file and the source data base files. These locations must be of the following form:

Host “username password”::Device:[Directory]

If the user presses RETURN at this point, the current default directory is searched for the file. The user has the option to abort the process by entering a “-1”.

Subroutine MERGE

This is a command routine that allow the user to merge an existing subsystem in a different case into the present case. The case name and subsystem name are input by the user as command arguments. The source case definition file is opened and all pertinent data are loaded into the present case data. Component names between the source components and the present case components are checked for duplication and the appropriate action is taken. The equipment data base files are updated by creating new data base entries for the merged components.

Functions and subroutines referenced:

CDCODE	CLOSE	CTOLOWERC	CTOUPPERC	DELREC
DUPLICATE	EQOPEN	FINDRM	FRAME1	FULL
INDEX	ISTAT		OPEN	OPENDB_X
RANDIN	RANDOU		RESTOR	SAVE
SETPRIM	TEK_ADV			

6.4 System Utility Routines

There are many system utility routines invoked by component routines to accomplish common tasks. These routines perform duties such as computing property values, interpolate data, check convergence on iteration loops, update output files and internal array values, etc. These routines are described below.

Subroutine BENCH(M,X)

This routine updates the component benchmark data in the CON array.

Functions and subroutines referenced: NONE

Subroutine BIVAR(X,Y,A,Z)

Performs an interpolation on a bivariate array (A).

Functions and subroutines referenced: INTER

Subroutine CDEL(NEA,NEB,ISTR)

This routine is used to check convergence for the mass.

Functions and subroutines referenced: ABS DPCS

Subroutine CDELA(NEA,NEB,ISTR)

This routine records the stream and constituent with the highest change from the last iteration value for each accumulator (TANKS, CABINS, SUMS, and NODES only).

Functions and subroutines referenced: ABS

Subroutine CILLCHAR(CISC,LEN,CHRFLG)

This routine parses names such as component names, filenames, case names, etc., to find illegal characters in the names.

Functions and subroutines referenced:

CHAR ICHAR TEK_ADV

Subroutine CONINIT

This routine is used at the beginnng of the solution to initialize the output CON location.

Functions and subroutines referenced: NONE

Subroutine CONV(X,Y,NC,CRELAX,XLAST,YLAST)

This routine is a convergence iteration routine used for implicit solutions.

Functions and subroutines referenced: NONE

Subroutine CORRECT(NC)

This routine adjusts the mass fraction of a connection to sum to 1.0. Corrects round off errors.

Functions and subroutines referenced: NONE

Subroutine CTOLOWERC(CCHAR,NCHAR)

This subroutine converts the character array CCHAR of NCHAR characters from uppercase characters to lowercase.

Functions and subroutines referenced: ICHAR

Subroutine CTOUPPERC(CCHAR,NCHAR)

This subroutine converts the character array CCHAR of NCHAR characters from lowercase characters to uppercase.

Functions and subroutines referenced: ICHAR

Subroutine DEL(IREC)

This routine is used to delete the last record written to the data base.

Functions and subroutines referenced:

ABS LBIT RANDIN RANDOU SETPRIM

Subroutine DENVIS(NEA,NSA,NEB,NSB,DEN,VIS)

This routine returns the average density (DEN) and viscosity (VIS) for a flow between components NEA and NEB in the stream connected by NSA and NSB.

Functions and subroutines referenced: NONE

Function DEWPT(PARTIAL_PRESS)

This routine is used to determine the dewpoint of air based on the water vapor partial pressure.

Functions and subroutines referenced:

INTER TEK_ADV

Subroutine DINTER(X,A,Y)

This routine performs an interpolation of single arrays with all of the arguments in double precision.

Functions and subroutines referenced: NONE

Subroutine DIST (IX1, IY1, IX2, IY2, XD)

This routine returns the sum of the squares, XD, of the horizontal and vertical distances between the coordinates (IX1, IY1) and (IX2, IY2).

Functions and subroutines referenced: NONE

Subroutine DUPLICATE(IREC,JREC)

This routine is used to duplicate a record in the data base.

Functions and subroutines referenced:

FINDRM	INDEX	OPENDB_X	RANDIN
RANDOU	RESTOR	SETPRIM	

Subroutine ERRDUMP(ITEST)

This routine displays error messages on the screen.

Functions and subroutines referenced: DPCS TEK_ADV

Subroutine FINDC(IEQ)

This routine is used to load the NPT array with pointers to the correct “C” array location for the component being solved.

Functions and subroutines referenced: SMVBITS

Subroutine FLOLEG(XM,DEN,VIS,XL,XD,XK,PDEL,MAX)

This routine is used to calculate flow conductance for resistance and flow information by iteration upon press drop “PDEL”.

Functions and subroutines referenced: FRICTDP

Subroutine GIMAG(IEQB,ISTB,NCPT,PDEL,MAX,GIX)

This routine calculates the flow conductance “GI” for stream “ISTRM” for a given pressure drop “PDEL” and mass flow rate. It is utilized in several flow balancing components.

Functions and subroutines referenced:

CONV	DENVIS	FRICTDP	LOG10

Subroutine INTER(X,A,Y)

This routine performs an interpolation of single arrays.

Functions and subroutines referenced: TEK_ADV

Subroutine KHECK(NUSTM)

This subroutine calls the convergence check routines for all of a component’s streams.

Functions and subroutines referenced:

CDEL	CORRECT	PDEL	TDEL

Subroutine KHECKA(NUSTM)

This routine performs the same function as KHECK but does not increment the LPCS count.

Functions and subroutines referenced:

CDELA CORRECT PDEL TDELA

Subroutine LOADCOND(IEQA,ISTA,IEQB,ISTB,XM,DP)

This routine loads pressure drop (DP) and flow rate (XM) into the “FLOCOND” array.

Functions and subroutines referenced: TEK_ADV

Subroutine MASSFRAC(NUM,YI,XMWI,XI)

This subroutine accepts an input number of constituents (NUM), an array of constituent mole fractions (YI) and an array of constituent molecular weights (XMWI) and calculates a corresponding array of mass fractions (XI).

Functions and subroutines referenced: NONE

Subroutine MODBAK

This routines creates a backup model file CASENAME.BAK.

Functions and subroutines referenced:

CLOSE CTOLOWERC INDEX OPEN
TEK_ADV

Subroutine MOLEFRAC(NUM,PRESSI,YI)

This subroutine accepts an input number of constituents (NUM) and an array of constituent pressures (PRESSI) and calculates a corresponding array of mole fractions (YI).

Functions and subroutines referenced: NONE

Subroutine PASSIVE(NSTM,NSI,NSO)

This routine checks to see if a passive outlet stream, NSO, is connected to an active inlet stream, NSI.

Functions and subroutines referenced: DPCS SCALER

Subroutine PDEL(NS)

This routine determines component relaxation of the pressure calculations.

Functions and subroutines referenced: ABS DPCS

Subroutine POINTCON

This routine determines the pointers (NINP,NOUT,NBEN) for the CON array locations for each component.

Functions and subroutines referenced: NONE

Subroutine PREPRO

This subroutine is used to convert OPS files from using the interface routines to using direct references.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CTOUPPERC	EQLOAD	FINDC
LIB$INDEX	PSEUDO	TEK_ADV		

Function ROWATER(TEMP)

This function determines the density of water vapor at a given temperature TEMP. This routine is located in PROPS.FOR.

Functions and subroutines referenced: NONE

Subroutine PULLSTX

This routine is used by the controller routine to pull the memory stack down one location. It is located in CNTRLLR.FOR.

Functions and subroutines referenced: NONE

Subroutine PUSHSTX

This routine is used by the controller routine to push the memory stack up one location. It is located in CNTRLLR.FOR.

Functions and subroutines referenced: NONE

Subroutine QEXCHG(IQ,FAE,GCONV,GCOND,TRAD,TCONV,TCOND)

This subroutine calculates the heat exchange between components and the assigned cabin environment.

Functions and subroutines referenced: TBOUND

Subroutine RBIVAR(X,Y,A,Z)

This routine is a bivariate interpolation routine that works in reverse to BIVAR .

Functions and subroutines referenced: INTER

Subroutine RINTER(Y,A,X)

This subroutine allows interpolation from a single array in reverse order of subroutine “INTER”.

Functions and subroutines referenced: TEK_ADV

Subroutine SAVEAS(NAME)

This routine creates a new case from the current case by duplicating all the data base entries with the new casename NAME. It also creates a new MOD file. The new case becomes the default.

Functions and subroutines referenced:

DUPLICATE	EQOPEN	FRAME1	ISTAT	MODBAK
RANDIN	RANDOU	SAVE		TEK_ADV

Subroutine SCALE(IST)

This routine is used to convert the mass fractions in the C array to mass flow rate and back to fractions. The routine has two ENTRY points:

ABS	SCALEUP(IST)	SCALEDN(IST)

It is called first during a component routine as SCALEUP(IST) so that computations can be done by mass flow rate. At the end of the routine it is called as SCALEDN(IST) to convert back to mass fractions. IST is the stream number to be "scaled."

Functions and subroutines referenced: NONE

Subroutine SCALER(ISTRM,NSTRM,DESFLW,ALWFLOW)

This routine is used to scale the requested flowrate from a store device to the available quantity for that iteration.

Functions and subroutines referenced:

ABS	DPCS	FINDC

Subroutine SUMINIT

This routine initializes the summary arrays for the simulation wrap up.

Functions and subroutines referenced: NONE

Subroutine TBOUND(IQ,TRAD,TCONV,TCOND)

This subroutine retrieves the environmental temperatures for the components.

Functions and subroutines referenced: NONE

Subroutine TDEL(NS)

This routine determines the relaxation values for temperature for a given component.

Functions and subroutines referenced: ABS DPCS

Subroutine TDELA(NS)

This routine is the same as TDEL but used for a given stream (used in the STORE and CABIN logic).
Functions and subroutines referenced: ABS

Subroutine TIMESTEP(TSTEP,N,GSUM,CAPAC,TAU,MINILPS)

This routine determines a time constant TAU and number of subtime steps (MINILPS) for a system of N diffusion nodes (N=10 max).

Functions and subroutines referenced: NONE

Subroutine TSTEP(A,B,C,D)

This routine is used to calculate the largest difference between A-C and B-C.

Functions and subroutines referenced: NONE

6.5 Model Archive Routines

The archive process (new in version 5.0) provides the user with the ability to store CASE/A simulation solution files to a specified directory. These files can be retrieved for later use. There are two steps to archiving a model: (1) edit the archive data base to describe general information about the archived models, (2) initiate the archive process to save the data to an archive file. Routines called to support step 1 are described in section 3.2, Interactive Editing, of this manual. Routines that support step 2 are described next.

Subroutine ARCHIVE

This routine stores the currently loaded CASEA model data to an ASCII file and is called directly from CASEAMAIN upon entry of the ARCHIVE command. The routine prompts the user for the name of the model that is to be archived and other pertinent information. If the user continues, the archive directory is determined and plotsets, ops code, .MOD file, or whatever is selected by the user to be archived is copied to this directory. If the archive process succeeds, a message stating “ARCHIVE OPERATION FINISHED” is written to the terminal and control is returned to CASEAMAIN.

Functions and subroutines referenced:

ARCHFILE	CTOLOWERC	EQOPEN	MERGE_OUT
RANDIN	TEK_ADV		

Subroutine ARCHFILE

This routine is called by ARCHIVE to check for active archive sets in the IA array. If more than one is active, the user is warned and only the first set is used. If none are found, a warning prompt is given to the user and the routine returns.

Functions and subroutines referenced: TEK_ADV

Subroutine RETRIEVE

This routine is invoked when the user enters the RETRIEVE command. It provides the user with a method to retrieve a previously archived file. This routine first prompts the user for the directory and file containing the archive file, then calls MERGE_IN to read the archive file data into the currently loaded model.

Functions and subroutines referenced:

ANSWER CTOLOWERC
GET_SET_DDIR (contained in RETRIEVE) MERGE_IN

SECTION 7. USER OPERATIONS LOGIC AND INTERNAL CASE/A DATA ACCESS

CASE/A provides for user-written computational logic at strategic points during the simulation. This capability is similar to that of programs like SINDA (VARIABLES1, VARIABLES2) and G189A (GPOLY1, GPOLY2). CASE/A provides seven such opportunities in the subroutines OPS0 to OPS6. Additionally the subroutine OPS7 is provided during component simulation as a "BLACKBOX". These program modules, referred to as "OPS LOGIC", provide for such things as custom variable initialization, time-dependent forcing functions, specialized output, etc. The use of OPS logic is presented in detail in chapter 7 and examples of use are provided in models WCM and MB6 in appendix A of the CASE/A User's Manual. This section provides a brief introduction to OPS logic and a guide to link user OPS logic with the CASE/A library.

7.1 OPS Logic Description

- OPS0, PREINITIALIZATION—This routine is called prior to execution of the component 100 block (data initialization). It is useful for initializing custom output variables, opening output files, and changing values contained in the data base prior to their use in the 100 block.

- OPS1, PRESIMULATION—This routine is similar in nature to OPS0, however, it executes just after the component 100 block. Thus, this routine is useful for initializing data that depends on initial conditions of the components.

- OPS2, PRETIME STEP—This subroutine is especially useful for doing time-dependant computations to impose time-variant conditions on the system. OPS2 is called every time step just prior to iteration.

- OPS3, INTERNAL SYSTEM ITERATION—The OPS3 logic executes every system iteration. It proves most useful in debugging when the user wishes to examine data or print diagnostic messages during the simulation. It is also used as a tool for performing conditional tests for feedback control.

- OPS4, POSTTIME STEP—Executed just after convergence of the system is obtained, but prior to incrementing the time, this block is useful for updating output files and performing integration functions.

- OPS5, OUTPUT INTERVAL—This routine is executed after OPS4 but only at the specified output interval. It is useful for performing operations that should coincide with the system output interval.

- OPS6, SIMULATION END—This routine is called after the simulation end time is reached. It is used to perform post sim output user wrap-up operations such as closing files.

7.2 Creation of User OPS Logic

OPS logic should be created in the user's directory (in the current working subdirectory). A logical approach to this is to first copy the file OPS.FOR from the CASEA$CODE directory to the user's directory under the name of the current case:

```
$copy casea$code:ops.for disk:[user.directory]casename.for
```

The next step is to edit this file with a text editor and code the logic required. Even if no code is placed into a subroutine, it is important that each subroutine be left intact, since the subroutines are

called regardless of whether they perform any function or not. An example of this completed OPS logic might look as follows:

```
       SUBROTINE OPS0
C  PRE-SIMULATION USER-DEFINED CODE
       RETURN
       END
C
C
      Subroutine OPS1
      INCLUDE 'SYS$CASEA:COMPCOM.INC'
C
C  OPS1 LOGIC - PRE SIMULATION.  The OPS1 subroutine for this case
C  is used for initialization of data parameters and resequencing
C  of the solution procedure.
C
      COMMON /ONE/ THW,TSHWR,TDWSH,TLDRY
      COMMON /TWO/ NHW,NSHWR,NDWSH,NLDRY
      COMMON /THREE/ DT1,DT2,DT3,DT4
      COMMON /FOUR/ IFLG,JFLG,KFLG,LFLG
      COMMON /FIVE/ TSTRT1,TSTRT2,TSTRT3,TSTRT4
      COMMON /SIX/ TSTOP1,TSTOP2,TSTOP3,TSTOP4
      COMMON /SEVEN/ NCREW,TFLG1,II,ICTRH,ICTRS,ICTRD,ICTRL
      COMMON /TEST/ SRCFLO,KFLAG
      COMMON /FLORATS/ F1SET,F2SET,F3SET,F4SET
      CALL GETU(1,THW)
      CALL GETU(2,TSHWR)
      CALL GETU(3,TDWSH)
      CALL GETU(4,TLDRY)
      CALL GETU(5,XNHW)
      CALL GETU(6,XNSHWR)
      CALL GETU(7,XNDWSH)
      CALL GETU(8,XNLDRY)
      CALL GETU(9,DT1)
      CALL GETU(10,DT2)
      CALL GETU(11,DT3)
      CALL GETU(12,DT4)
      CALL GETU(13,XIFLG)
      CALL GETU(14,XJFLG)
      CALL GETU(15,XKFLG)
      CALL GETU(16,XLFLG)
...
      CALL SEQUENCE('PS3     ')
      CALL SEQUENCE('S4      ')
      CALL SEQUENCE('PR4     ')
      CALL SEQUENCE('DSHWSH  ')
      CALL SEQUENCE('PR5     ')
      CALL SEQUENCE('LDRY    ')
      CALL SEQUENCE('SEQSTOP ')
     CALL SETK('S1      ',11,0.5)
      CALL SETK('S1      ',12,0.5)
      CALL SETK('S2      ',11,0.5)
      CALL SETK('S2      ',12,0.5)
      CALL SETK('S3      ',11,0.5)
      CALL SETK('S3      ',12,0.5)
      CALL SETK('S4      ',11,0.5)
      RETURN
      END
C
C
C
      Subroutine OPS2
      INCLUDE 'SYS$CASEA:COMPCOM.INC'
C
C  OPS2 LOGIC - BEFORE EACH TIME STEP.  The OPS2 subroutine will be
```

```
C  used to control the flow of hot water at the handwash, shower,
C  dishwash and laundry facilities according to the timeline described
C  in section A.3.1.
C
      COMMON /ONE/ THW,TSHWR,TDWSH,TLDRY
      COMMON /TWO/ NHW,NSHWR,NDWSH,NLDRY
      COMMON /THREE/ DT1,DT2,DT3,DT4
      COMMON /FOUR/ IFLG,JFLG,KFLG,LFLG
      COMMON /FIVE/ TSTRT1,TSTRT2,TSTRT3,TSTRT4
      COMMON /SIX/ TSTOP1,TSTOP2,TSTOP3,TSTOP4
      COMMON /SEVEN/ NCREW,TFLG1,II,ICTRH,ICTRS,ICTRD,ICTRL
      COMMON /TEST/ SRCFLO,KFLAG
      COMMON /FLORATS/ F1SET,F2SET,F3SET,F4SET
C
      IF (ICTRH .NE. 4) THEN
        IF (TIME .GE. TSTRT1 .AND. IFLG .EQ. 1) THEN
          IFLG   = 0
        ENDIF
        IF (TIME .GE. TSTOP1 .AND. IFLG .EQ. 0) THEN
          IFLG   = 1
          TSTRT1 = TIME + DT1           ! set start time for next
          TSTOP1 = TSTRT1 + THW         ! crew members handwash
          II     = II + 1               ! during the present cycle
          IF (II .EQ. NCREW) THEN
            TSTRT1 = TFLG1 + 4.0        ! begin the next handwash cycle
            TSTOP1 = TSTRT1 + THW       ! 4 hours after the beginning
            TFLG1  = TFLG1 + 4.0        ! of the previous cycle
            II     = 0
            ICTRH  = ICTRH + 1
          ENDIF
        ENDIF
      ENDIF
.
.
.
.
        SPLT2L2 = FLO12/FLOTOT          ! split frac of leg 2 of S2
        CALL SETK('S2      ',11,SPLT2L1)
        CALL SETK('S2      ',12,SPLT2L2)
C
        SPLT4L1 = FLO4/FLO34            ! split frac of leg 1 of S4
        SPLT4L2 = FLO3/FLO34            ! split frac of leg 2 of S4
        CALL SETK('S4      ',11,SPLT2L1)
        CALL SETK('S4      ',12,SPLT2L2)
C
        SPLT1L1 = FLO1/FLO12            ! split frac of leg 1 of S1
        SPLT1L2 = FLO2/FLO12            ! split frac of leg 2 of S1
        CALL SETK('S2      ',11,SPLT2L1)
        CALL SETK('S2      ',12,SPLT2L2)
      ENDIF
C
      RETURN
      END
C
C
C
      Subroutine OPS3
C
C  OPS3 LOGIC - EACH ITTERATION
c
c *** NOTE THAT THIS Subroutine SHOULD BE INCLUDED IN CASEANAME.FOR
c *** EVEN THOUGH IT DOESNT DO ANYTHING
C
30    RETURN
      END
C
```

```
C
C
      Subroutine OPS4
      INCLUDE 'SYS$CASEA:COMPCOM.INC'
C
C  OPS4 LOGIC - AFTER EACH TIME STEP.  The OPS4 subroutine will be
C  used for adjustment of certain parameters associated with the
C  STORE component (WTRTNK).  Output parameters of interest for this
C  case are evaluated graphically using the Integrated Plot Utility.
C
      COMMON /TEST/ SRCFLO,KFLAG
C
C
      CALL GETK('WTRTNK  ',19,WMASI)    ! initial tank contents mass
      CALL GETK('WTRTNK  ',73,WMAS)     ! current tank contents mass
C
.
.
.
.
      TL    = SV*4.0/3.14159
      GWALL = 0.3996*TL + 0.157
      AO    = 5.3668*TL + 1.704
      GCONV = 0.2*AO
      GRAD  = 0.1714E-8*AO*0.05*0.9
C
      CALL SETK('WTRTNK  ',12,GCONV)    ! reset tank convection, radiation
      CALL SETK('WTRTNK  ',13,GRAD)     ! and wall conductor conductance
      CALL SETK('WTRTNK  ',15,GWALL)    ! values
C
      RETURN
      END
C
C
C
      Subroutine OPS5
C
C  OPS5 LOGIC - EACH OUTPUT INTERVAL
c
c *** NOTE THAT THIS Subroutine SHOULD BE INCLUDED IN CASEANAME.FOR
c *** EVEN THOUGH IT DOESNT DO ANYTHING
C
      RETURN
      END
C
C
C
      Subroutine OPS6
C
C  OPS6 LOGIC - POST SIMULATION
c
c *** NOTE THAT THIS Subroutine SHOULD BE INCLUDED IN CASEANAME.FOR
c *** EVEN THOUGH IT DOESNT DO ANYTHING
C
      RETURN
      END
C
C
C
      Subroutine OPS7
C
C  OPS7 LOGIC - BLACKBOX SIMULATION
C
      RETURN
      END
```

A complete description of this problem is in appendix A of the User's Manual.

The next step is to compile the program and link with the rest of the CASE/A object library. It is important to note that compilation and linking must be done every time the OPS Logic file is modified to effect a change in the simulation. This procedure is accomplished as follows:

```
$ fortran/nooptimize/nodebug/check=bounds/continue=45 casename.for
$ @casea$code:linkcas casename
$ run/nodebug casename
```

This procedure will compile the code and give the option to use the debugger if required. LINKCAS.COM links with the object code located in the directory CASEA$CODE and creates an executable CASENAME.EXE. The nodebug qualifier should be left out of the RUN statement if the user wishes to use the VMS debugger. It is beyond the scope of this manual to present the use of the debugger. LINKCAS is described in the User's Manual.

Subroutine OPS0

OPS0 logic is called by the SOLVE routine prior to the component 100 block.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS1

OPS1 logic is called by SOLVE prior to simulation but after the component 100 block.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS2

OPS2 logic is called prior to each time step by SOLVE.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS3

OPS3 logic is called by SOLVE each iteration until convergence.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS4

OPS4 logic is called by SOLVE after convergence but before time is incremented.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS5

OPS5 Logic is called by SOLVE after OPS4 but only when the time is a multiple of the output interval SYSOUTPT.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS6

OPS6 Logic is called by SOLVE after the end of a simulation to allow for postsimulation wrap-up.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Subroutine OPS7

OPS7 Logic is called by SOLVE to simulate a blackbox component defined by the user.

Functions and subroutines referenced:

Any CASE/A or user-written subroutine can be called.

Normally the user will create a subroutine for the blackbox component and call this routine in OPS7 logic when OPS7 is called by SOLVE (for example, see example 2 – MB6 in the CASE/A User's Manual section 1).

7.3 Case/A Internal Data Communication Arrays

It may be useful for the user to check or vary the value of "internal" data associated with the simulation. There are five main arrays that CASE/A uses to communicate data to and from the different routines in the program. These arrays, and the subroutines used to manipulate them, are described here and in chapter 7 of the User's Manual. These very important data structures, which are used extensively in modeling systems with CASE/A, are the CON, C, PRO, USERCON, and D arrays. The subroutines used to store and retrieve data are described in section 7.3.6.

7.3.1 The "CON" Array

The CON array can be thought of as a "LIST" of component data for every component in the simulation (fig. 17). This "LIST" is comprised of 75,000 "CELLS" grouped into SEGMENTS. Each SEGMENT contains all of the data for a component. For each simulation, there are NEQ (number of active pieces of equipment) SEGMENTS. The remaining portion of the array is unused. The relative equipment number IEQ is used throughout the simulation to refer to a SEGMENT in this array for a component.

Each segment is divided into four contiguous BLOCKS:

(1) Fixed Block contains nine fixed parameters that exist for every component. LFXD in the lower part of figure 16 is a constant set to nine, which refers to the length of this block.

The next three blocks are variable in length for every component type:

(2) Input Block contains the INPUT data for the component, and its length is determined by the variable LINP.

(3) Output Block contains the OUTPUT data and its length is determined by the variable LOUT.

(4) Benchmark Block contains the BENCHMARK data, and its length is determined by the variable LBEN.

Thus, the total length of each segment would be LFXD+LINP+LOUT+LBEN.

The beginning of each block of data is determined by an integer pointer to the first cell of the block. These pointers are NFXD, NINP, NOUT, and NBEN, which point to the absolute location in the CON array for the FIXED, INPUTS, OUTPUTS, and BENCHMARKS blocks, respectively. These pointers are calculated in the POINTCON routine as follows:

```
NFXD = ICP(IEQ) - 1
NINP = NFXD + LFXD
NOUT = NINP + LINP
NBEN = NOUT + LOUT
```

where: `ICP is an array of pointers to the beginning location in CON array for each component segment`

```
LFXD = 9
LINP = CON(NFXD+1)
LOUT = CON(NFXD+2)
LBEN = CON(NFXD+3).
```

Figure 17 shows the arrangement of the CON array. Also, refer to chapter 7 of the User's Manual.

7.3.2 The "C" Array

The C array, or composition array, is used to track the fluid mass flow rate and composition for each of the component fluid streams. This 54 by 1,500 array can track up to 49 constituents for 1,500 connecting streams. The first index of the C array is for flow rate and composition and the second is for the stream connection number. The C array is graphically shown in figure 18.

Location (1, "stream number") of the C array is always mass flow rate in lbm/h. The next 49 locations are for mass fractions of the fluid constituents in the stream of interest (these fractions should sum to 1.0). The first eight of these constituents are defaults shown in table 2. The variable NTRACK (not contained in the C array) is equal to the index location of the last constituent tracked. Thus, if the defaults only are used, NTRACK would be equal to nine. The next four locations, NTRACK+1, NTRACK+2, NTRACK+3, and NTRACK+4, have special meanings for several routines and should not be used arbitrarily.

If the user needs to track compounds other than the eight default, several steps are required. First, the PROPS routine should be modified to obtain the properties of that compound. This new PROPS routine would then be linked with the rest of the CASE/A library in the same manner as OPS logic. The CONTROLS data base should be edited next and the value of NTRACK changed from 9 to the total constituents to be tracked (plus one for mass flow rate). Finally, the LABELS data base can be edited to associate a name with that location of the C array.

Most components require a number of C locations equal to the number of inlet and outlet streams. For example, a PUMP requires two C locations, one for the inlet and one for the outlet (as shown for IEQB in fig. 18). Other components, however, require additional C locations to account for

mass accumulation and other processes. For instance, the FILTER component uses four C locations—one for the inlet, one for the outlet, and two to keep track of matter accumulating on the filter media. In general, a component may have up to 8 streams (refer to components in section 10 of the User's Manual). Each stream is referenced in the component routine by the array NPT. For example, NPT(3) refers to stream 3 of the component currently being processed. The value contained in NPT(3) is the index into the C array for that stream.

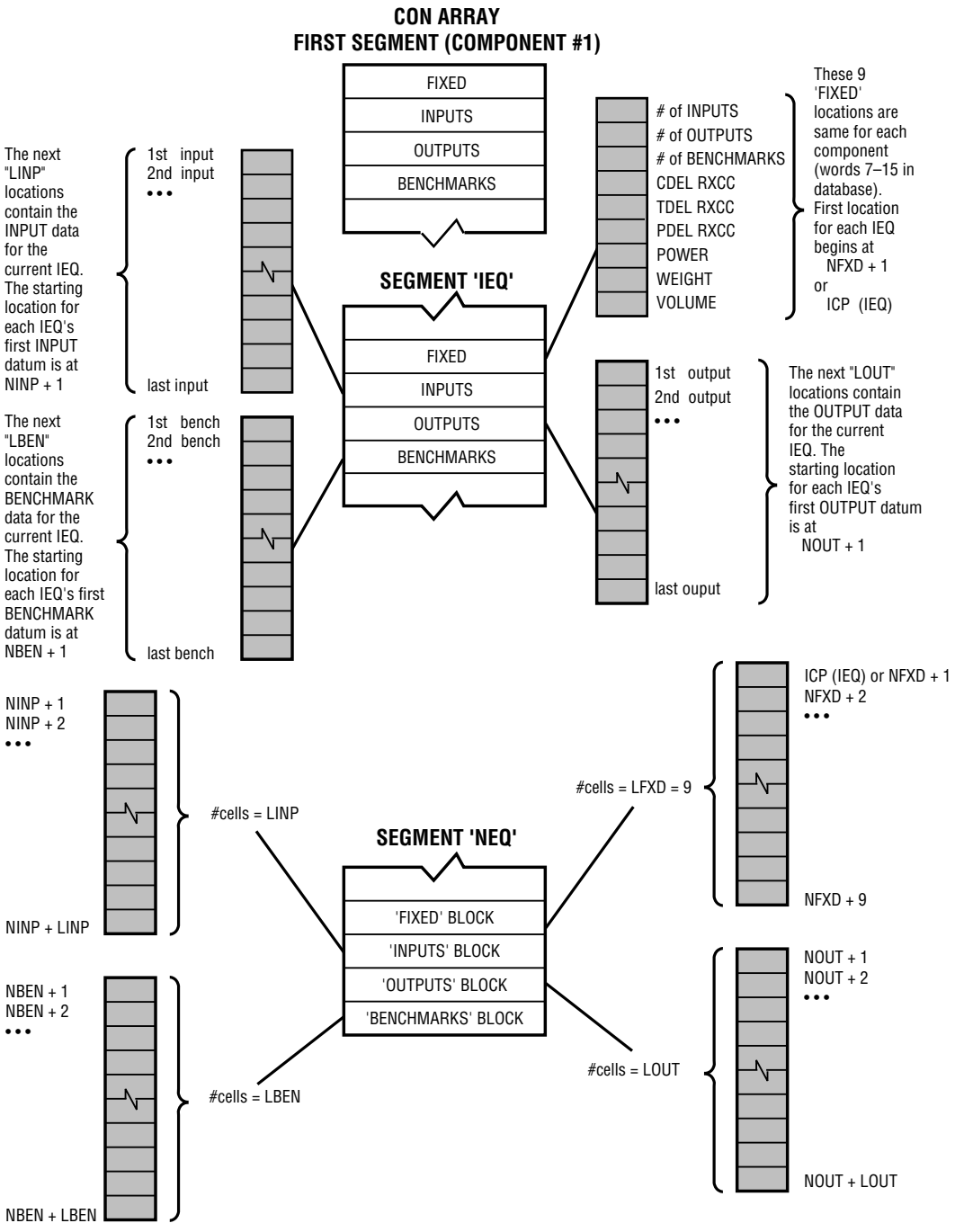


Figure 17. The "CON" array.

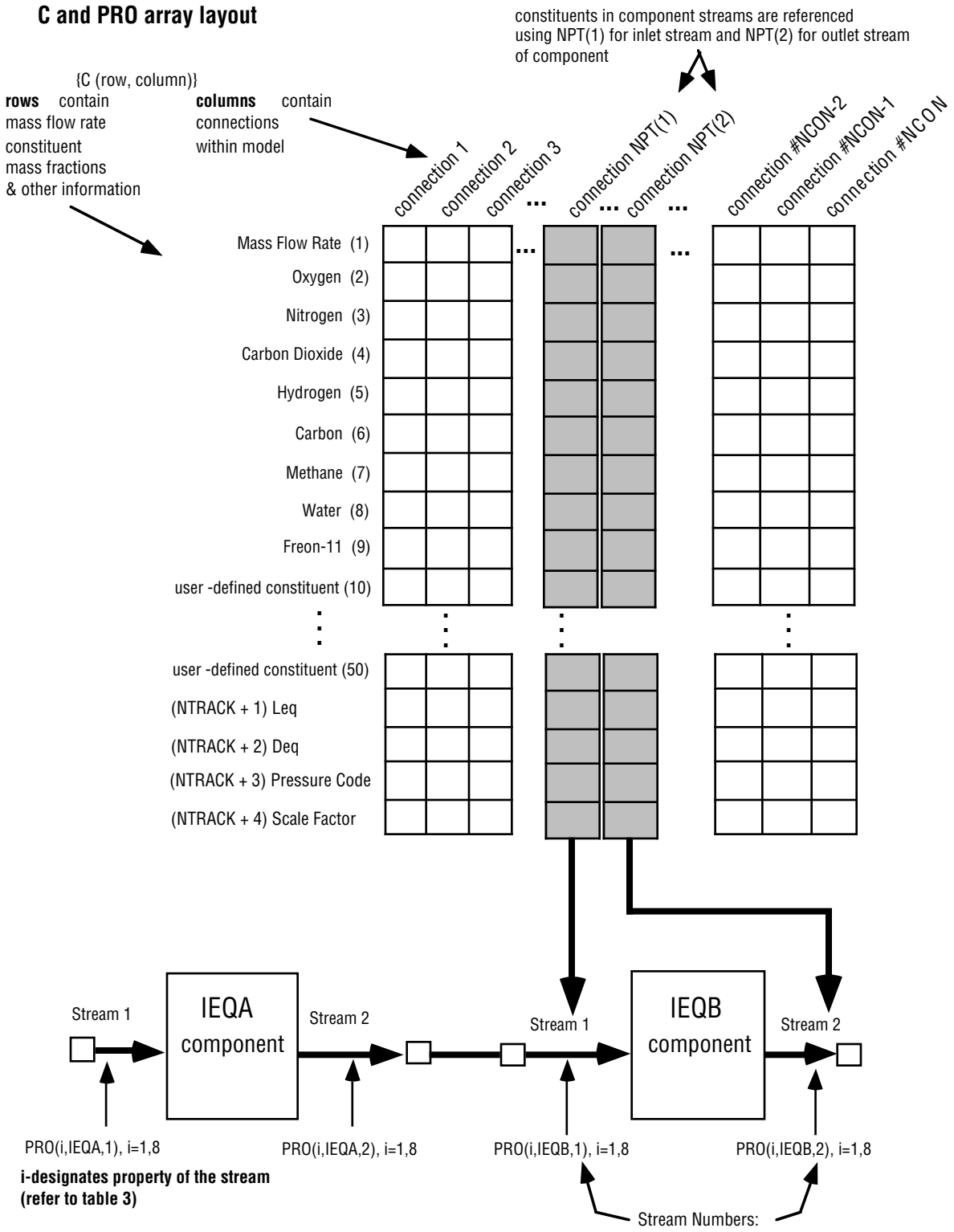


Figure 18. Stream properties and composition arrays.

Table 2. "C" array default constituents.

LOCATION	CONTENT	Units
1	Total Mass Flow Rate	Lbm/Hr
2	Oxygen	Mass Fraction
3	Nitrogen	Mass Fraction
4	Carbon Dioxide	Mass Fraction
5	Hydrogen	Mass Fraction
6	Solid Carbon	Mass Fraction
7	Methane	Mass Fraction
8	Water	Mass Fraction
9	Freon - 11	Mass Fraction
10	Constituent 10	Mass Fraction
11	Constituent 11	Mass Fraction
12	Constituent 12	Mass Fraction
.	.	.
.	.	.
.	.	.
50	Constituent 50	Mass Fraction
NTRACK+1	Equivalent Length	Feet
NTRACK+2	Equivalent Diameter	Inches * 100
NTRACK+3	Pressure Feedback Code	
NTRACK+4	Scale Factor	

The subroutine PSEUDO is used to keep track of the pointers that give access to the C array. The subroutines GETC and SETC are available to store and retrieve data in this array.

7.3.3 The "PRO" Array

The PRO array stores the thermodynamic properties for each component stream (refer to the bottom of figure 18). This three-dimensional array can track up to 12 properties for 1,000 components with up to 8 streams each using the first, second, and third indexes, respectively. The properties represented by the first index value are shown in table 3. Note that there is room for future expansion or user-defined properties since the array is dimensioned to 12 and only 8 properties are tracked. The routines SETP and GETT are used to store and retrieve data from this array.

Table 3. "PRO" array properties list.

LOCATION	CONTENT
1	Pressure, lb/in^2 absolute
2	Temperature, Fahrenheit
3	Specific Heat (Cp), Btu/(lbm°F)
4	Density, lbm/ft^3
5	Viscosity, lbm/(h·ft)
6	Specific Heat (C_v), Btu/(lbm°F)
7	Molecular Weight, lbm/lb-mole
8	Enthalpy, Btu/lbm

7.3.4 The “USERCON” Array

CASE/A provides an additional array that is carried in the common block /USERCON/ in CASEA$CODE:GRAPHCOM.INC. This one-dimensional array can be used to store data and communicate it throughout the simulation. Up to 100 different parameters can be tracked with this array. Each location can be associated with a label by editing the USERCON data base.

This array is a one-dimensional single precision real array and can be accessed directly or with the GETU and SETU routines. Upon completion of the simulation, the final values of each location are stored in the data base. Thus, this array should be initialized by the user in OPS0 logic for each new simulation unless data accumulation is the desired result.

7.3.5 The “D” Array

The D array dimensioned to D(1000,50) is a system level array used in the component subroutines. The first index is the relative equipment number (IEQ) for a component in a given case. The second index represents 50 available locations for “random” use. Many components use the D array. Each routine should be examined to determine what if anything is stored in locations 1 to 50. The purpose of this array is to provide access to variables internal to a component routine that are not otherwise available to the user (i.e., through the component edit screen). After careful checking that an array location is free, an experienced user is encouraged to exploit this array to access internal calculations of a component. Note that to reference the array one should use the form D(IEQ,i) where i is the desired location and IEQ is set by the system in the SOLVE routine.

7.3.6 Storage and Retrieval Functions for CASE/A Arrays

The subroutines used to store data into and retrieve data from the CON, C, PRO, and USERCON arrays are described below.

Subroutine GETC(NAME,ISTR,ICONST,VALUE)

This routine retrieves a value from the C array. It returns the VALUE of the ICONST constituent number of the ISTR stream for the component NAME.

Functions and subroutines referenced: TEK_ADV FINDC

Subroutine GETI(NAME,IEQVAL)

This routine returns the relative equipment number (IEQVAL) for the component NAME. It can be used in conjunction with the GETK routine to get the value of ICON.

Functions and subroutines referenced: TEK_ADV

Subroutine GETK(NAME,ICON,VALUE)

This routine is used to retrieve data from the CON array. NAME is a CHARACTER*8 variable (INTEGER*4(2)). The routine returns the VALUE of the ICON relative location for the component NAME.

Functions and subroutines referenced: TEK_ADV

Subroutine GETP(NAME,ISTR,VALUE)

This routine returns the VALUE from the PRO array of the pressure of stream ISTR for the component NAME.

Functions and subroutines referenced: TEK_ADV

Subroutine GETPP(NAME,ISTR,ICONST,PP)

This routine returns the partial pressure for the equipment name and stream and user-specified constituent ICONST.

Functions and subroutines referenced: TEK_ADV FINDC

Subroutine GETT(NAME,ISTR,VALUE)

This routine is similar to GETP but returns temperature.

Functions and subroutines referenced: TEK_ADV

Subroutine GETU(ILOC,VALUE)

This is used to get a value from the USERCON array. It returns the VALUE of the ILOC location.

Functions and subroutines referenced: TEK_ADV

Subroutine SETC(NAME,ISTR,ICONST,VALUE)

This routine is functionally opposite to GETC in that it stores a value in the C array.

Functions and subroutines referenced: TEK_ADV FINDC

Subroutine SETK(NAME,ICON,VALUE)

This routine is the opposite of GETK in that it stores a VALUE in the CON array.

Functions and subroutines referenced: TEK_ADV POINTCON

Subroutine SETP(NAME,ISTR,VALUE)

This routine is functionally opposite to SETP in that it stores a value in the PRO array.

Functions and subroutines referenced: TEK_ADV

Subroutine SETT(NAME,ISTR,VALUE)

This routine is functionally opposite GETT. It stores a temperature in the PRO array.

Functions and subroutines referenced: TEK_ADV

Subroutine SETU(ILOC,VALUE)

This routine is used to set the ILOC location of the USERCON array to VALUE.

Functions and subroutines referenced: TEK_ADV

Subroutine PREPRO

This routine is the OPS logic preprocessor discussed in section 7.5 of the User's Manual. The preprocessor helps speed execution by eliminating dependence on the GET and SET routines discussed above. GET and SET routines search from 1 to NCOMP until finding the correct equipment name before retrieving or depositing data. The preprocessor optimizes this process by writing OPS code to access the correct array (C, PRO, CON) location directly.

Functions and subroutines referenced:

CDCODE	CILLCHAR	CTOLOWERC	CTOUPPERC
FINDC	FOR$CLOSE	FOR$OPEN	LIB$INDEX
PSEUDO	TEK_ADV		

SECTION 8. ANALYTICAL TECHNIQUES

This section discusses the techniques used to perform hydraulic and thermal computation. The major subroutines used in these computations are described briefly along with the functions and subroutines called by them.

8.1 System Pressure Computations

CASE/A provides two methods of computing pressure at key points in the system. These two methods are toggled by the variable MATRXFLG from the CONTROL data base.

8.1.1 Matrix Reduction Pressure Solution

The default method of determining the system pressures is the solution of a set of simultaneous equations by matrix reduction. This set of equations, discussed in section 8.2.1 of the User's Manual, is obtained by conservation of mass at each pressure node. The mass flows and compositions are set by the component routines, while the pressures are calculated by the system using the routine SYSBAL.

The overall hydraulic balance usually requires several iterations in that the components make successive "guesses" at the flow rates based on the system calculated pressures. The linearized flow conductor between two nodes is thus a function of the flow rate and pressure difference occurring between them.

The system solution methodology is based on the concept of the subdivision of the entire hydraulic layout of the case into separate subnetworks that are terminated by boundary pressure nodes or sources and sinks. These subnetworks may be connected by shared boundary pressure nodes (i.e., STORE's and CABIN's). A set of simultaneous equations is set up and solved for each of these subnetworks individually. This is done in order to decrease the computational resources that would be required to solve a large set of equations for all nodes in the case. The matrix solution used is the symmetric Cholesky method derived from SINDA/SINFLO. Each of the subnetworks should have at least one boundary pressure or a singular matrix condition occurs in that no solution is possible. In this case, an arbitrary node in the network will be taken as a boundary node and the rest of the pressures calculated accordingly. The user is notified with a run-time warning if this situation occurs.

Subroutine SYSBAL

This routine is responsible for solving for the system pressures given flow rates and conductor values as described above and in the User's Manual. The routines used by this routine are not discussed here, but are documented in the file SYSBAL.FOR.

Functions and subroutines referenced:

CONCALC	DPOFA	DPOSL	GFINIT
PSPCFD	SETFLO	TEK_ADV	

8.1.2 Hydraulic Solution

The hydraulic solution system is responsible for the pressure drop and mass flow balance versus flow resistance calculations on both the component and system levels. The hydraulic solution is based on an implicit solution method that relies on feedback of the downstream pressures to the upstream components through the flow path connections. This feedback approach allows the characteristics of a

closed flow path to be sensed by the component, which initiates the mass flow while it is in direct communication only with the immediately adjacent components. Thus, no knowledge of the complete flow layout is needed, and the mass pressure balance calculations are made by the components that initiate the mass flows or that split the flow based on the downstream flow resistances. A few system iterations are usually required to achieve the correct pressure balance, but the flow configuration and PCS are the determining factors as to how fast the system converges. An optimized PCS causes a very significant increase in execution speed, since fewer iterations are required if the components are called in a head-to-tail sequence. The user may alter the PCS through the use of the SEQUENCE routine in the user operations blocks. This routine allows the user to specify a group of components in a sequential list as they are to be called in that relative order. The rest of the PCS is not affected. Of course, the user may rearrange the whole PCS if necessary by listing all of the components in the case in the desired order. The user should place the SEQUENCE calls in the OPS1 block if an initial sequence is desired or in the OPS2 block if time dependent resequencing is desired.

The hydraulic solution system consists of component-level routines that calculate internal component pressure losses, as well as system-level routines that perform the interface operations between the component streams through the flow path connections. The interface routines base their operations on the stream classification codes discussed in the next section. Two of the routines that perform the interface operations are COPY and COPYA, which are discussed below.

Subroutine COPY(NUMSTM,CFLAG)

This routine updates stream composition and flow rate (C Array) data to each component. It is called at the beginning of the 200 block in the component subroutines. Properties (PRO Array) are also updated for all inlet stream to the given component.

Functions and subroutines referenced:

CONDP	CORRECT	DPCS	FINDC
PROPS			

Subroutine COPYA(NUMSTM)

This routine is similar to COPY.

Functions and subroutines referenced:

CORRECT	FINDC	PROPS

8.1.3 Stream Classifications

The component streams are classified as one of six general types of flow “devices” according to their mass flow characteristics. There are four general categories of stream types: inlet, outlet, conditional, and stagnant. There are two types of input streams, two types of output streams, and one each of the conditional and stagnant types:

1. Passive inlet: Indicates that the stream accepts whatever mass flow is passed to it by the upstream component. An example of this type is the inlet stream of a filter. Its pressure is determined by adding the pressure drop of that component’s internal flow path to the corresponding outlet stream pressure of the same component. Thus, a feedback loop is established between the inlet stream and its corresponding outlet stream.

2. Active inlet: Indicates that the stream generates a flow rate by pulling mass from the upstream component. An example of this type is the inlet stream of a

pump or fan. Its pressure is set equal to the upstream pressure minus the pressure drop through the flow connection.

3. Passive outlet: Indicates that the component passes the flow from a passive inlet stream through to the outlet side with the establishment of a feedback loop between the inlet side and the outlet side. Therefore, its pressure is set equal to the downstream component's inlet pressure plus the pressure drop through the flow connection. An example of this type is the outlet stream of a filter.

4. Specified outlet: Indicates that the component contains some type of flow generation device internal to the component that drives the outlet flow without a direct relationship to any of the inlet streams. Its pressure may be set to a constant value to simulate a regulated process, or a feedback loop may be established by passing the PSPEC routine an argument value of zero. In the latter instance, the pressure will be adjusted to match the downstream flow configuration. An example of the regulated type is the oxygen outlet stream of the static feed water electrolysis (SFWE), which relies on a high-pressure nitrogen source to regulate the reaction chamber pressure. An example of the feedback type is a component that contains an integral positive displacement pump.

5. Conditional: Indicates that the stream may either accept flow or pass flow depending on (1) what is hooked to it or (2) on the pressure balance of the system. The component routine is responsible for any interface operations or feedback loop establishment and the calculation of the stream pressure. Examples of this type are tanks and cabins.

6. Stagnant: Indicates that no flow can occur to or from the stream no matter what the flow conditions are. This type usually indicates a deactivated component or perhaps a closed valve.

These stream classifications are contained in the system level data array named HYDRA and may be changed dynamically during the solution process as conditions dictate by the component routines. These stream codes are initialized to the component default values at the beginning of the simulation in the routine PINIT. They are checked for compatibility in the CONDP interface routine, that is responsible for calculating the pressure drops in the flow connections and performing feedback operations, so that the connections make sense from a hydraulic standpoint. For instance, the connection of two input streams to each other is an obvious error and a warning message is printed to the display. In this case, the simulation is continued, but the pressure drop for the flow connection is not calculated and the pressures are left at their previous values. The user may encounter a situation where dynamic conditions cause a few solution iterations in that the stream codes are incompatible, but that eventually sort themselves out. However, a repetitive series of the warning messages indicates an erroneous connection that the solution system cannot resolve.

8.1.4 Friction Losses Through Connections

The pressure drop through each of the flow connections is calculated by the routine FRICTDP, which is called by the routine CONDP mentioned in the previous section. The FRICTDP routine calculates the pressure drop based on the assumption of an incompressible fluid flowing through a smooth circular conduit in the laminar, transitional, and turbulent regimes. The Darcy formula is used in combination with the hydraulic equivalent length and diameter, the fluid density, and the fluid viscosity to calculate the pressure drop. Note that the dynamic pressure losses due to geometric considerations must be accounted for by the correct calculation of the equivalent length for the specified equivalent diameter. These values are stored in the constituent "C" array and are loaded at the beginning of the simulation

from the CASENAME.MOD file, that contains the graphical information of the system layout. The user must specify the equivalent length and diameter when the CoNect command is issued in the buildup of the system layout (CN;Leq;Deq). The length, Leq, is entered in feet and the diameter, Deq, is entered in hundredths of an inch (100 = 1 inch). For instance, a connection that is 25-feet long and 6 inches in diameter should be entered as CN;25;600. Both values must be entered only as integers, since the internal storage array may contain only integers. If no arguments are included for the CN command, the system automatically sets the equivalent length and diameter to their default values of 1 foot and 6 inches, respectively.

Subroutine CONDP(NEA,NSA,NEB,NSB,NC)

This routine determines the pressure drop between components NEA and NEB connected by streams NSA and NSB whose connection number is NC.

Functions and subroutines referenced:

DENVIS FRICTDP LOADCOND TEK_ADV

Subroutine FRICTDP(SDEN,SVIS,SXL,SXD,XMDOT,DPX)

This routine calculates the pressure drop, DPX, through a smooth pipe of a diameter, SXD, and equivalent length, SXL, for a flow with the properties of density = SDEN and viscosity = SVIS at a rate of XMDOT.

Functions and subroutines referenced: NONE

8.1.5 Pressure Loss Through Components

The component routines are responsible for the calculation of the pressure losses for any internal flow paths. The system-level routines in turn use these values to establish the feedback loops when required. The routine PIPEDP is called by most of the components that have simple flow-through fluid paths. It calls the routine FRICTDP, described in the previous section, with the appropriate equivalent length and diameter, which are usually contained in the component performance data in the CON array.

The routine COMPDP is called by those components that make specialized pressure drop calculations, but also need to establish feedback loops with the rest of the system. This routine accepts the internally calculated pressure drop as an argument and sets the component stream pressures accordingly. For example, the FILTER routine must adjust the pressure drop across the filter as it gets clogged with debris, and this calculation is specific only to the filtration process. However, a feedback loop is still required with the rest of the system to obtain the correct pressure balance. Therefore, the FILTER routine calculates the pressure drop across the filter element and then calls the COMPDP routine to carry out the feedback function.

The routines that simulate the components that redirect or split the flow contain specialized logic to carry out these functions. They are responsible for their own interface operations as well as maintaining the feedback loops. An example of this type is the NODE component routine. It is the most complicated routine from a hydraulic standpoint, since it must determine the flow directions as well as the flow rates of a variable number of connecting streams. It relies on the connecting stream pressures, connection flow path resistances, and connecting stream classifications to arrive at the correct flow balance.

The components that have “specified outlet” streams call the routine PSPEC to set their outlet pressures or establish feedback loops as appropriate. This routine is used for those outlet streams that are regulated to a set pressure or that correspond to the outlet side of an internal positive displacement flow device (constant flow rate with variable pressure differential). A nonzero value for the pressure

establishes the stream as a regulated pressure stream, while a value of zero establishes a feedback loop to match the stream's pressure to the downstream conditions.

Subroutine COMPDP(NSTM,NSI,NSO,DP)

This routine sets the outlet pressure of the stream NSO based on the inlet pressure of stream NSI and the pressure drop DP.

Functions and subroutines referenced: LOADCOND

Subroutine PIPEDP(NSTM,NSI,NSO,XL,D)

The subroutine, similar to COMPDP, computes the outlet pressure of stream NSO based on the inlet stream pressure NSI using the equivalent length XL and equivalent diameter D.

Functions and subroutines referenced:

FRICTDP LOADCOND PROPS

Subroutine PSPEC(NSTR,PRESS)

This routine sets the outlet pressure for stream NSTR of component IEQ to its specified value of PRESS.

Functions and subroutines referenced: NONE

8.2 Thermal Network Solution Routines

The solution methodology used to solve for unknown system temperatures is discussed in section 8.3 of the User's Manual. Many components use the generic thermal network discussed below.

Subroutine TNETWK(MAX, RELAX, TSTEP, G1, G2, G3, G4, G5, FAE, G7, G8, CAP,QMASS, QSHEL, TRAD, TCONV, TCOND, TIN, TMI, TOUT, TWAL, TMAS, TSHEL, IC)

This routine solves a generic thermal network discussed in section 8.3 of the User's Manual and is used by many of the components for the thermal solutions.

Functions and subroutines referenced: MTH$EXP

Subroutine TNETWK2(MAX, RELAX, TSTEP, G1, G2, G3, G4, G5, FAE, G7, G8, CAP,QMASS, QSHEL, TRAD, TCONV, TCOND, TIN, TMI, TOUT, TWAL, TMAS, TSHEL, IC)

This routine is the same as TNETWK except the FT term that approximates the mass average temperature is not used.

Functions and subroutines referenced: MTH$EXP

8.3 Mass Transfer

Subroutine MNETWK

MNETWK performs mass transfer network calculations for components modeling a fluid stream in contact with a sorbent bed (SAWD, MOLSIEV, DEFLOW, etc.) The delta network technique is used as described in section 8.5 of the User's Manual.

Functions and subroutines referenced:

BIVAR	RBIVAR	SUBR	SUBR2

8.4 Thermodynamic Properties

Thermodynamic properties, such as enthalpy, specific heat, etc., are calculated for each fluid stream by the PROPS routine. PROPS is set up for the eight default constituents discussed in section 7.2.2. These properties are stored in the PRO array as discussed in section 7.2.3. Constituents other than the eight defaults are assumed to have the properties of water unless the PROPS routine is modified to reflect the new constituent (see section 7.2.2).

Subroutine PROPS(NE,NS,NC)

Calculates the properties of a mixture based on the mass weighted average values of the constituents' properties at the specified pressure (lb/in^2 absolute) and temperature ($^{\circ}F$), given in the PRO array, and composition (mass fractions contained in the "C" array).

Functions and subroutines referenced:

INTER	SATPR	TEK_ADV	VISC
USERPROPS			

Function SATPR(TEMP)

Returns the saturation pressure of air at specified temperature.

Functions and subroutines referenced:

MTH$ALOG	MTH$EXP	RINTER

Subroutine VISC(C,W,T0,XMU0,T,XMU)

Used to determine the viscosity of a fluid.

Functions and subroutines referenced: NONE

SECTION 9. COMPONENT ROUTINES

CASE/A presently supports 53 components. From the predefined components and the additional "BLACKBOX" component (OPS7), the user may construct a system to simulate many life support system configurations. Additionally, an experienced programmer can develop custom models to add to the CASE/A component library. Each type of component is associated with a type number, usually referred to by the variable ITYPE. ITYPE is contained in the third location of the IEL array for each component in the model (i.e., ITYPE = IEL(IEQ,3)). These type numbers are shown in figure 19.

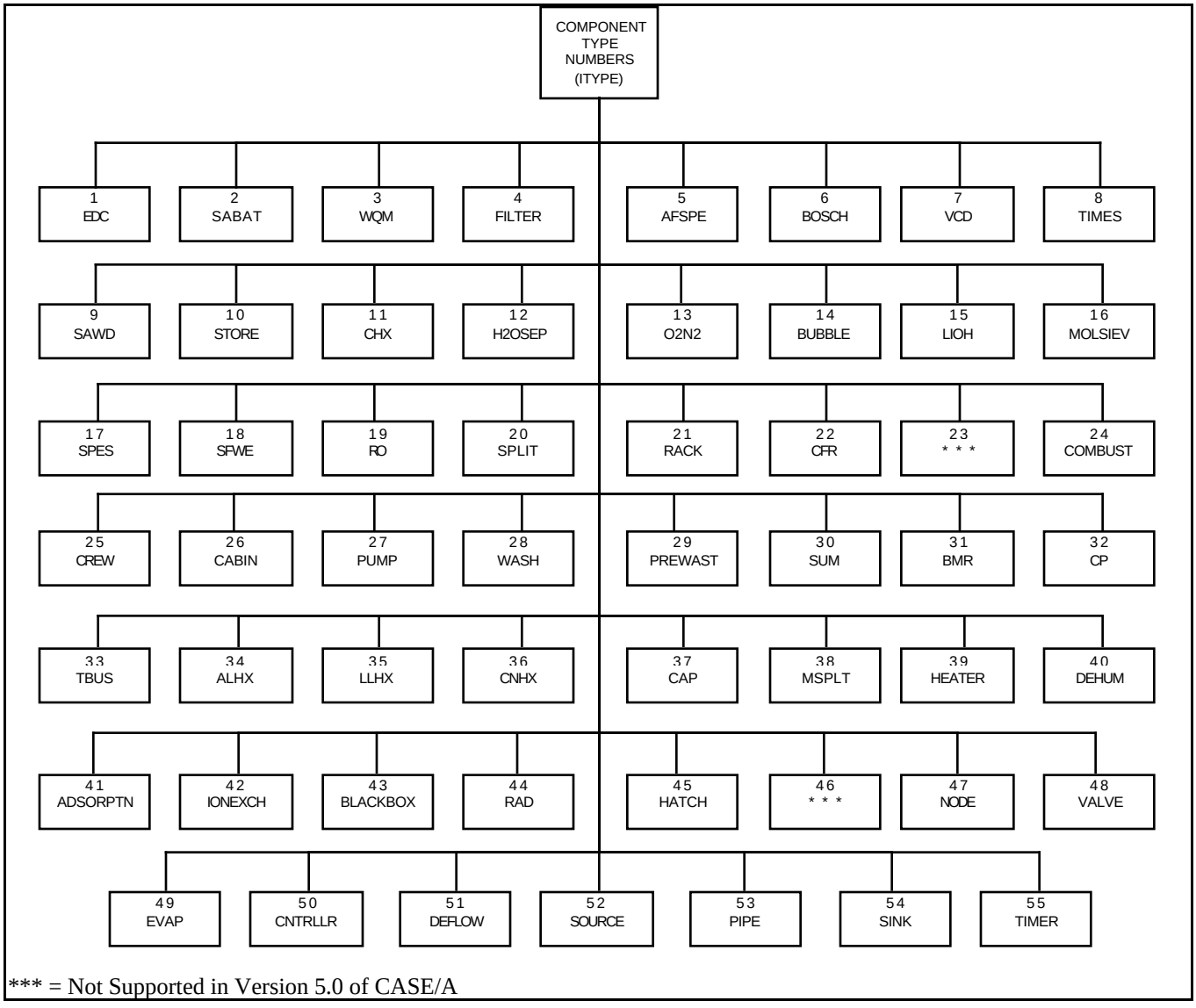


Figure 19. Component "ITYPES".

9.1 Component Routine Logic Structure

The CASE/A system presently supports 53 component types from which the user may construct a life support system model. A subroutine exists for each of these component types that is called from the SOLVE routine. Several unique components of the same type may be located in a configuration,

each with its own operating parameters, but the same subroutine is called for each. All component subroutines perform four discrete functions at the direction of the SOLVE routine via the system common block variable MFLAG (fig. 20). These functions correspond to the four segments of the simulation execution logic. The four functions are discussed below.

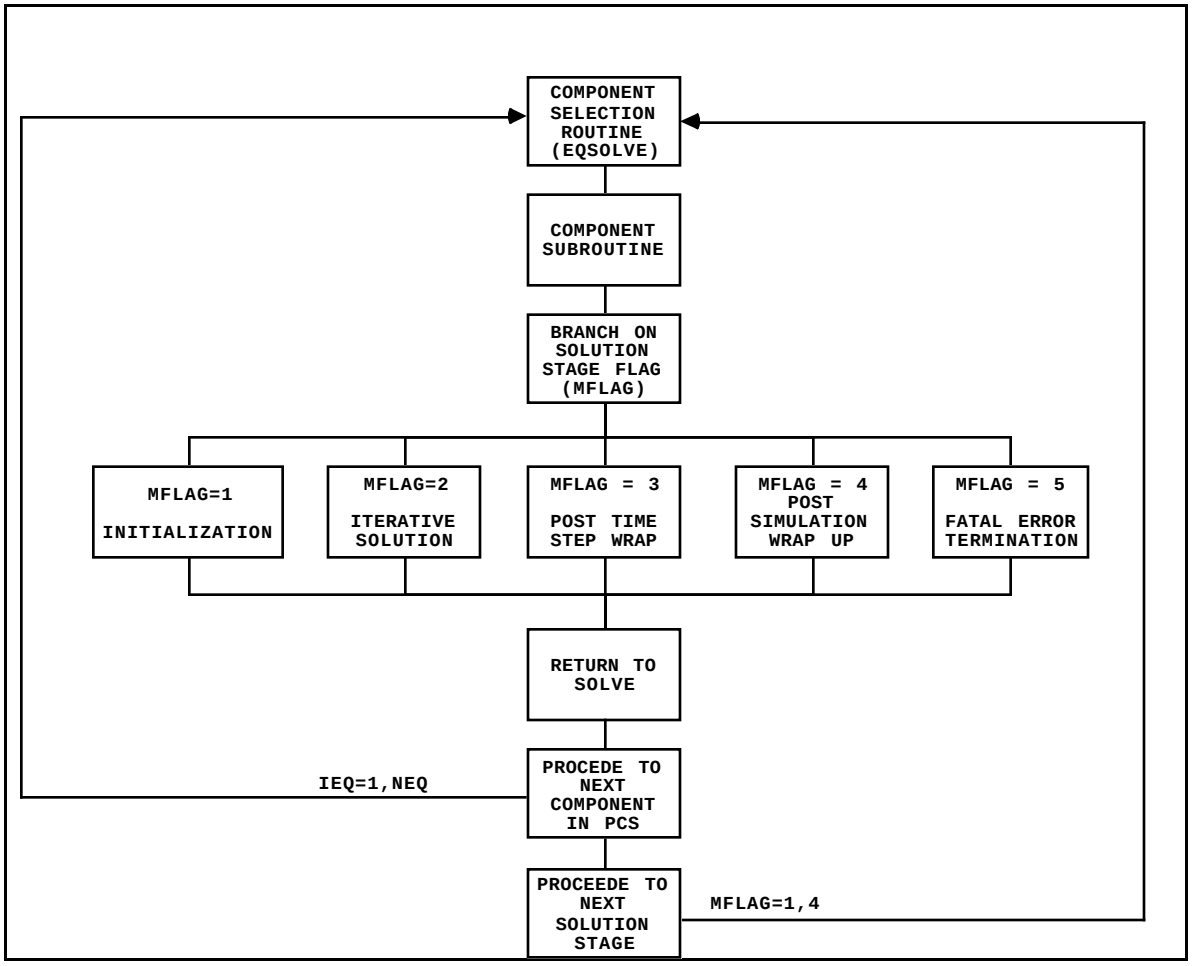


Figure 20. Component logic flow diagram.

9.1.1 Initialization Segment

When MFLAG is set equal to one, each component routine performs an initialization function that loads the component performance data into active memory from mass storage and performs any other operation required to initialize the component. This segment is executed only once during the course of the simulation for each component.

9.1.2 Iterative Solution Segment

When MFLAG is set equal to two, each component routine performs the operations associated with the simulation of that component type's physical behavior. The mass flow, thermal, and hydraulic calculations are performed according to the operating characteristics for each component in the case.

Component types can be classified as active or passive devices relative to mass flow considerations. Active devices will generate a flow rate: for example, a PUMP generates flow according to the rate specified by the user or by its characteristic curve. The mass flow of an active device is usually specified by the user, but it may be calculated by the routine as a consequence of its internal operation.

Passive devices accept the flow from the upstream component and operate appropriately on that mass flow. The output of a passive device is a function of the input flow plus or minus the internal losses or redirection of flow. A FILTER component is an example of a passive device. In some instances, a component can have a combination of both passive and active behavior. For example, the BOSCH routine, that simulates the BOSCH carbon dioxide reduction process, accepts concentrated carbon dioxide from an upstream component that may or may not be mixed with the required quantity of hydrogen. If the incoming mixture is deficient in hydrogen, the component will attempt to draw the makeup hydrogen from the component connected to the BOSCH hydrogen makeup stream.

Each component inlet stream "accepts" the flow conditions from the upstream component's outlet stream. The component routine performs its calculations based on these inlet flow conditions existing as fixed boundary values. The component routine then sets the outlet stream conditions according to the performance parameters and operating characteristics of that particular type of equipment. A convergence check on the mass flows, temperatures, and pressures of all streams is then performed and execution control is then returned back to the SOLVE routine. This segment is the main body of the component logic and is executed several times during the solution of each time interval until system convergence is obtained.

9.1.3 Posttime-Step Wrap-Up Segment

When the system has obtained convergence for the present time step, MFLAG is set equal to three and each component routine performs a post time step wrap-up where mass accumulation devices perform specialized calculations, and benchmark data, if tracked, are updated for all components. This segment is executed once for each time interval of the simulation.

9.1.4 Postsimulation Wrap-Up Segment

When the simulation has been completed, MFLAG is set equal to four and each component performs a post simulation wrap-up where the benchmark data are written back to the mass storage file for that component. This segment is executed only once at the end of the simulation.

9.1.5 Internal Fatal Error Condition

A mechanism for terminating the simulation from within a component routine has been incorporated into the solution system for those circumstances when a fatal error condition has been generated within a component routine. This is accomplished by setting the value of MFLAG to five in the component routine itself. There are conditional test statements in the SOLVE routine that check for this flag after each component routine returns control back to SOLVE. The SOLVE routine then returns control directly back to the main command processor without performing the simulation wrap-up. A description of the error condition will be written to the display device by the component routine.

9.2 Component Routines

The following is a brief description of each component subroutine including functions and subroutines referenced by each. A complete description of the component and its modeling assumptions is contained in chapter 10 of the User's Manual.

Subroutine ADSORPTN

This routine is used to simulate the ADSORPTioN units used in water recovery systems.

Functions and subroutines referenced:

BENCH	COMPDP	CONINIT	COPY CORRECT
EQWRIT	EXP	KHECK	PASSIVE
QEXCHG	TBOUND	TEK_ADV	TNETWK

Subroutine AFSPE

This routine is used to simulate the process of hydrogen and oxygen generation through the electrolysis of water by a solid polymer electrolysis unit.

Functions and subroutines referenced:

AFPSATW	AFSPEHX	BENCH	CONINIT
COPY	EQWRIT	PSPEC	SPE1
SPE2			

Subroutine BMR

This routine is used to simulate the body mounted radiator.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
EXP	KHECK	PASSIVE	PIPEDP
TEK_ADV			

Subroutine BOSCH

This routine simulates the BOSCH CO_2 reduction reactor.

Functions and subroutines referenced:

ALOG	BENCH	B_CHX	CONINIT
COPY	DEWPT	EQWRIT	KHECK
PASSIVE	PIPEDP	PSPEC	SQRT
TEK_ADV			

Subroutine CABIN

This subroutine is the primary routine to simulate CABIN module environments.

Functions and subroutines referenced:

BENCH	CONINIT	COPYA	CORRECT
DEWPT	DPCS	EXP	FINDC EQWRIT
KHECKA	PROPS	SATPR	TEK_ADV

Subroutine CAP

This routine simulates the thermal behavior of a phase-change thermal capacitor.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
EXP	KHECK	PASSIVE	PIPEDP
TEK_ADV			

Subroutine CFR

This routine is the primary routine used to simulate the carbon formation reactor. Used in conjunction with a Sabatier CO_2 reduction reactor, the carbon formation reactor converts methane from the Sabatier into solid carbon.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
FINDC	KHECK	PASSIVE	PIPEDP
PSPEC	TEK_ADV		

Subroutine CHX

This subroutine is the primary routine used to simulate the condensing heat exchanger. It is modeled after the condensing heat exchanger used on *Spacelab* and the shuttle.

Functions and subroutines referenced:

ABS	ALOG	BENCH	CONINIT	
COPY	DEWPT	EQWRIT	EXP	
KHECK	PASSIVE	PIPEDP	PSPEC	SATPR
TEK_ADV				

Subroutine CNHX

This subroutine is the primary routine used to simulate the contact heat exchanger.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
KHECK	PASSIVE	PIPEDP		

Subroutine CNTRLLR

This routine is the driver for simulating the controller component. The controller provides the user a means of changing system variables without using OPS logic. It is useful for providing feedback control loops for components and to simulate time varying component performance.

Functions and subroutines referenced:

ABS	ACOS	ALOG	ALOG10	
ASIN	BENCH	CDCODE	CONINIT	
COS	COSH	EQWRIT	EXP	
GETC	GETI	GETK	GETP GETT	
POINTCON	PULLSTX	PUSHSTX	SETC SETK	
SETP	SETT	SIN		
SINH	SQRT	TAN	TANH	TEK_ADV

Subroutine CP

This subroutine is the primary routine to simulate a cold plate heat exchanger.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
KHECK	PASSIVE	PIPEDP	QEXCHG	
TBOUND	TEK_ADV	TNETWK		

Subroutine CREW

This is the primary routine used to simulate the metabolic functions such as respiration, perspiration, food and water consumption, heat rejection, and waste production of a crewperson.

Functions and subroutines referenced:

ABS	COPY	EQWRIT	KHECK	PSPEC
SCALER	TEK_ADV			

Subroutine DEFLOW

This is the primary routine for simulating a desicant bed used in the molecular sieve for water vapor removal.

Functions and subroutines referenced:

ABS	BEDLOAD_DEFLOW	BENCH	CONINIT
COPY	DCONV	DINTER	DMNETWK
DTNETWK2	ETA_DEFLOW	EQWRIT	GETU
KHECK	LOOPWARN_DEFLOW	NUM_STEP_DEFLOW	PASSIVE
PIPEDP	PSPEC	SQRT	TBOUND
TEK_ADV	TIMESTEP	WARN_DEFLOW	

Subroutine DEHUM

This is the primary routine for simulating water vapor removal of a zeolite or silica gel dehumidifier for the molecular sieve.

Functions and subroutines referenced:

ABS	BENCH	BIVAR	CONINIT
COPY	EQWRIT	EXP	KHECK
PASSIVE	PIPEDP	PSPEC	SCALE
TEK_ADV	TSTEP		

Subroutine EDC

This is the primary routine used to simulate an electrochemical depolarized CO_2 concentrator (EDC) component. The EDC separates the metabolic carbon dioxide from the inlet air stream for delivery to a reduction unit such as the Bosch reactor.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	DPCS
EQWRIT	FINDC	KHECK	PASSIVE	
PIPEDP	PSPEC	SCALE	TEK_ADV	

Subroutine EVAP

This is the primary routine used to simulate a flash evaporator for removal of heat from a control fluid.

Functions and subroutines referenced:

BENCH	BIVAR	CONINIT	CONV
COPY	EQWRIT	FINDC	INTER
KHECK	PASSIVE	PIPEDP	TEK_ADV

Subroutine FILTER

This routine simulates the performance of a porous media filter.

Functions and subroutines referenced:

ABS	BENCH	COMPDP	CONINIT
COPY	CORRECT	EQWRIT	KHECK
PASSIVE	QEXCHG	TBOUND	TEK_ADV
TNETWK			

Subroutine H2OSEP

This routine is designed to simulate the H_2O separators used on board spacecraft and is modeled after the *Spacelab* version used in the *Spacelab* ECLS models. This component is used to remove the water from the condensing heat exchanger.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
KHECK	PIPEDP	PROPS	PSPEC	SATPR
TEK_ADV				

Subroutine HATCH

This is the primary routine to simulate the pressure equalization and air exchange functions of a hatch connecting two cabins.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPYA
DPCS	EQWRIT	FINDC	LOADCOND
PROPS	SCALER	TEK_ADV	

Subroutine HEATER

This routine simulates an ideal heater to elevate the temperature of a fluid stream.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
KHECK	PASSIVE	PIPEDP	TEK_ADV	

Subroutine HX

This routine is used by all of the heat exchanger components to simulate two fluid heat exchanges.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
EXP	KHECK	PASSIVE	PIPEDP	
TEK_ADV				

Subroutine IONEXCH

This routine simulates the performance of a cylindrical ion exchange unit.

Functions and subroutines referenced:

ABS	BENCH	COMPDP	CONINIT
COPY	CORRECT	EQWRIT	EXP
KHECK	PASSIVE	QEXCHG	TBOUND
TEK_ADV	TNETWK		

Subroutine LIOH

This routine simulates a lithium hydroxide (LiOH) cartridge. The LiOH cartridge is used to remove CO_2 from an air stream.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPY	EQWRIT
EXP	KHECK	PASSIVE	PIPEDP	
TBOUND	TEK_ADV	TNETWK		

Subroutine MODULE

Simulates a module that acts as a volume of containment to track atmospheric composition, mass addition/losses, and heat transfer to/from other modules or the external environment.

Functions and subroutines referenced:

ABS	BENCH	CONINIT	COPYA
CORRECT	DCONV	DPCS	EQWRIT
EXP	KHECKA	PROPS	TEK_ADV

Subroutine MOLSIEV

Simulates a MOLecular SEIVE CO_2 adsorption/desorption.

Functions and subroutines referenced:

BENCH	CONINIT	CONV	COPY
EQWRIT	FLOWSPACE	GET_CONTINUUM	
ISODAT	KHECK	MNETWK	MOL_ETA
PASSIVE	PIPEDP	PSPEC	RISODAT
SCALE	TBOUND	TEK_ADV	TIMESTEP
TNETWK2			

Subroutine MSPLT

Simulates the Multi-SPLiT or SPLiT component.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	DENVIS
DPCS	EQWRIT	FRICTDP	FLOWLEG
KHECK	LOADCOND	SCALER	TEK_ADV

Subroutine NODE

Simulates the NODE component. This routine is located in the file NODE1.FOR.

Functions and subroutines referenced:

COPYA	DENVIS	DPCS	EQWRIT	FINDC
FRICTDP	KHECKA	LOADCOND		
PROPS	SCALE	SCALER	TEK_ADV	

Subroutine O2N2

Simulates the nitrogen and oxygen partial pressure controllers.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
FINDC	SMVBITS	KHECK	PSPEC
TEK_ADV			

Subroutine OPS7

OPS7 logic blackbox component code.

Functions and subroutines referenced:

COPY	FINDC	KHECK	PDEL
SCALER			

Subroutine PIPE

Simulates the PIPE component.

Functions and subroutines referenced:

BENCH	COMPDP	CONINIT	COPY
DPCS	EQWRIT	FRICTDP	KHECK

PASSIVE TBOUND TEK_ADV TNETWK
QEXCHG

Subroutine PREWAST

Simulates a WASTE PREtreatment component.

Functions and subroutines referenced:

BENCH CONINIT COPY CORRECT
EQWRIT KHECK PROPS PSPEC
QEXCHG SCALER TBOUND TEK_ADV
TNETWK

Subroutine PUMP

Simulates the PUMP component.

Functions and subroutines referenced:

BENCH CONINIT CONV COPY
DPCS EQWRIT FINDC KHECK PSPEC
QEXCHG SCALER TBOUND TEK_ADV
TNETWK

Subroutine RACK

Simulates the RACK component.

Functions and subroutines referenced:

BENCH CONINIT COPY EQWRIT
KHECK MTH$EXP PASSIVE PIPEDP
TEK_ADV

Subroutine RAD

Simulates a RADiator component.

Functions and subroutines referenced:

BENCH CONINIT COPY EQWRIT
KHECK MTH$EXP PASSIVE PIPEDP
TEK_ADV TNETWK

Subroutine RO

Simulate a reverse osmosis component.

Functions and subroutines referenced:

BENCH COMPDP CONINIT CONV
COPY EQWRIT KHECK MTH$EXP
PASSIVE PROPS PSPEC QEXCHG
TBOUND TEK_ADV TNETWK

Subroutine SABAT

Simulates the SABATier component.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	DEWPT	
EQWRIT	KHECK	MTH$EXP	PROPS	PSPEC
PROPS	QEXCHG	SABCHX	SABDEL	
SABGCAL6	SABH2O	SABRAT	SCALE	
TBOUND	TEK_ADV	TEMPNET		

Subroutine SAWD

Simulates the solid amine water desorb unit.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	CONV	EQWRIT
ISOLCO2	ISOLH2O	ISOPCO2		ISOLPH2O
KHECK	LOOPWARN	MNETWK	PASSIVE	
PIPEDP	PSPEC	SAWD_ETA	TBOUND	
TEK_ADV	TNETWK	WARN		

Subroutine SFWE

Simulates the static feed water electrolysis unit.

Functions and subroutines referenced:

BENCH	BIVAR	CONINIT	COPY	EQWRIT
KHECK	PASSIVE	PIPEDP	PSPEC	SFWET
TBOUND	TEK_ADV			

Subroutine SINK

This routine simulates the SINK component.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
KHECK			

Subroutine SOURCE

This subroutine simulates an ideal fluid source, providing downstream components with a constant (or timelined in user OPS logic) input pressure, temperature, flowrate and composition.

Functions and subroutines referenced:

COPY	KHECK	PSPEC

Subroutine SUM

Simulates a component that mixes two fluid streams.

Functions and subroutines referenced:

COPYA	DENVIS	DPCS	ERRDUMP
EQWRIT	FINDC	FRICTDP	GIMAG
KHECKA	LOADCOND	PROPS	SCALE
SCALER	TEK_ADV		

Subroutine TANK

Used to simulate the STORE component.

Functions and subroutines referenced:

BENCH	CONINIT	COPYA	DPCS	EQWRIT
FINDC	KHECKA	PROPS	QEXCH	
TEK_ADV				

Subroutine TBUS

Simulates the thermal bus component.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	EQWRIT
KHECK	PASSIVE	PIPEDP	

Subroutine TIMER

This routine simulates the TIMER component.

Functions and subroutines referenced:

POINTCON	TEK_ADV

Subroutine TIMESC

Used to simulate a thermally integrated membrane evaporation system (TIMES component).

Functions and subroutines referenced:

BENCH	CONINIT	COPY	CORRECT
EQWRIT	KHECK	PSPEC	TEK_ADV

Subroutine VALVE

Simulates a VALVE component.

Functions and subroutines referenced:

BENCH	COMPDP	CONINIT	CONV
COPY	DPCS	EQWRIT	FRICTDP
KHECK	LOADCOND	PASSIVE	POINTCON
QEXCHG	TBOUND	TEK_ADV	TNETWK

Subroutine VCD

Simulates a vapor compressed distillation component.

Functions and subroutines referenced:

BENCH	CONINIT	COPY	CORRECT
EQWRIT	KHECK	PSPEC	TEK_ADV

Subroutine WASH

Used to simulate a WASH component.

Functions and subroutines referenced:

BENCH	COMPDP	CONINIT	COPY
CORRECT	EQWRIT	KHECK	PASSIVE
PROPS	PSPEC	QEXCHG	SCALER
TBOUND	TEK_ADV	TNETWK	

Subroutine WQM

Used to simulate a water quality monitor component.

Functions and subroutines referenced:

COPY	EQWRIT	KHECK	PASSIVE
PIPEDP	QEXCHG	TBOUND	TEK_ADV
TNETWK			

APPENDIX A. PREPARATION OF COMPATIBLE COMPONENT SUBROUTINES

The following sections describe how a user would incorporate a CASE/A compatible component subroutine that matches the existing program architecture. The process is broken down into three sections including graphical icon construction, data management initialization, and component routine construction.

1.0 Graphical Component Icon Construction

The CASE/A graphical component icon usually has one, two, or three entering and exiting streams, but may have more depending on the process at hand. Each entering and exiting stream is numbered from one to the total number of streams for the given component, with the entering streams being numbered first. Integer numbers are used to represent the individual ECLSS components and to specify their interconnections, order of solution, and the appropriate CASE/A subroutine that is assigned to simulate their function. Each piece of equipment located by the user will be assigned an arbitrary equipment number (IEQ) depending on the order of component input. The solution routine solves the component matrix by taking the component routines in the order of equipment input, with the exception of certain "flagged" components that are sorted to the beginning of the solution scheme.

Several routines, arrays, and variables must be changed in order to incorporate a new graphical icon. These routines work together in order to produce the graphics representation on the terminal screen. The following steps outline the sequence for inserting a new component icon into the existing data base.

1.1 Step 1: Increase Number of Components

The variable "NCOMP" refers to the active number of components that are readily available for a given ECLSS simulation. When making an addition to the current library, this variable must be increased by the number of components that are being added to the CASE/A library. The variable initialization occurs within the main program section of CASEAMAIN.FOR.

1.2 Step 2: Modify "Drawc" Routine

The majority of work involved with the location of a new component icon is contained in the routine DRAWC. When first entering this routine, the programmer finds a large number of array declarations all of the form "ICOMP(I)", "JCOMP(J,K)". The names ICOMP and JCOMP are shortened names for integer arrays containing data points for the representation of a particular component icon. For example, the arrays "IMOL" and "JMOL" contain the data points for graphical representation and rotation data for the MOLSIEV component. The values contained in the "ICOMP" array pertain to the coordinates of the graphical picture of the new component. The "JCOMP" array contains the coordinates data for the stream labels and component name.

The following example code in figure A-1 and figure A-2 use the MOLSIEV component to describe the "ICOMP(I)" and "JCOMP(J,K)" array usage. Note that coordinates referenced are relative to the center of the icon, unless otherwise noted, and are measured in pixels.

```
DATA IMOL   /60, 60, 8, -30, 0, 30, 0, 15, 30, -27, -30/

Data Items: 1, 2  =     Height x Width of box (60 x 60 pixels)
               3  =     number of remaining elements used in array
            4, 5  =     (x,y) coordinates of stream 1
            6, 7  =     (x,y) coordinates of stream 2
            8, 9  =     (x,y) coordinates of stream 3
          10, 11  =     (x,y) coordinates of body hit dot

Note that IMOL is dimensioned to 11


DATA JMOL   /-58,   5,  35,   5, -16, 42,  -for   0 degrees
             -26, -50, -26,  42, -61,  0,  -for  90 degrees
              38, -12, -55, -12, -10,-45,  -for 180 degrees
               5,  40,   5, -45,  35, -5/  -for 270 degrees
```

Figure A-1. Example code for new component in DRAWC routine.

Note that the MOLSIEV component has three stream labels. Therefore, three data pairs (six items) are required to locate the starting point of the label for each possible icon orientation (0˚, 90˚, 180˚, or 270˚).

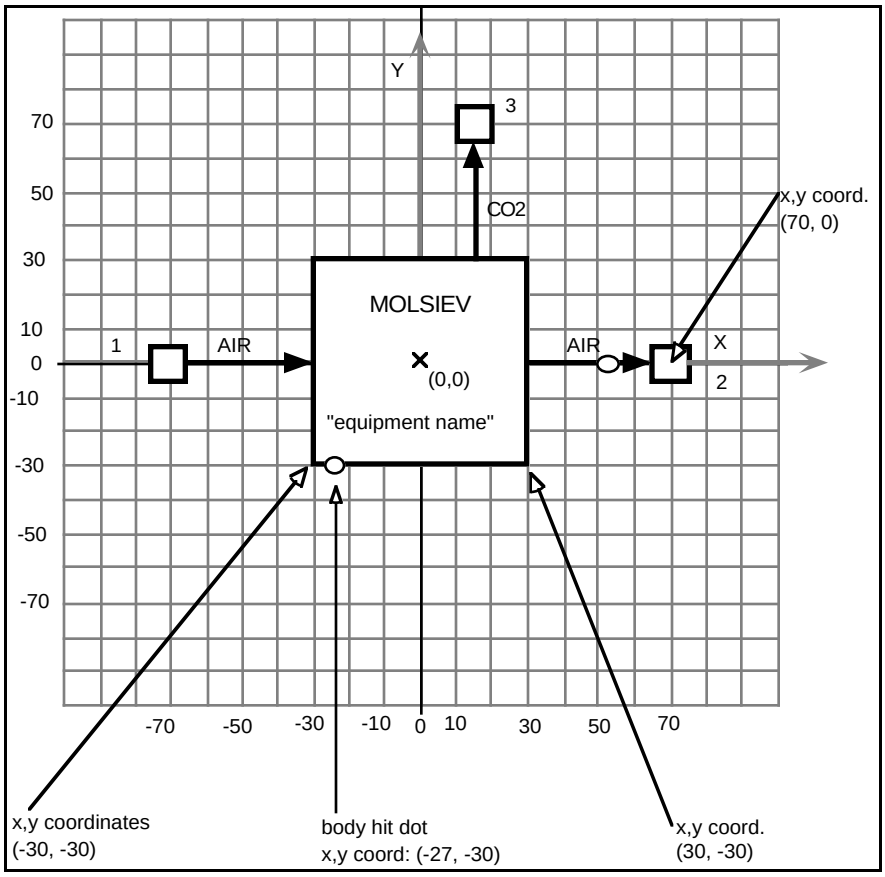


Figure A-2. MOLSIEV icon layout.

The next array to modify is NSCODE. This array is dimensioned to NSCODE (65, 8) and contains the stream code for each of the eight possible streams for every component type. Stream codes are discussed in section 8.2.3 of the user's manual and section 8.1.3 of this manual. They are summarized below:

CODE	TYPE
1	Passive inlet
2	Active inlet
3	Passive outlet
4	Specified (active) outlet
5	Accumulator/boundary stream
6	Stagnant stream

Since the MOLSIEV is type 16, its stream codes are stored in NSCODE(16,i), i=1,8. The MOLSIEV has three streams: stream 1 is a passive inlet, 2 is a passive outlet, and 3 is a specified outlet. The scream codes are therefore 1, 3, 4, 0, 0, 0, 0, 0. Zeros must be used for non-existent streams. For a new component, the programmer must specify the stream code of each inlet/outlet stream based on its function. For example, the MOLSIEV is a simple flow-through device. Air laden with CO_2 enters stream 1 and passes through a sorbent bed and exits stream 2 due to upstream/downstream drivers. Thus, the inlet and outlet streams are passive. When the bed is in desorb mode, a vacuum pump (internal to the component) draws flow through stream 3, thus, making stream 3 a specified (or active) outlet. These code types are default types only and an experienced programmer can develop a code that can change the stream types dynamically based on operating conditions.

The final modification to the DRAWC routine involves the code responsible for actually drawing the icon on the terminal screen. As described in section A.1.1, the first new component added to CASE/A will be component type 56. The DRAWC routine uses a computed GOTO statement to transfer control to statement label XX00 where XX is the component type number. Therefore, the code for component type 56 begins at statement label 5600. The following is a discussion of the MOLSIEV (ITYPE=16) icon code that begins at statement label 1600. This segment of code is shown in figure A-3.

ASIDE: If the new component icon could be represented by the MOLSIEV, the programmer could simply copy the block of code starting at label 1600 and ending before label 1700 to a new section beginning with label 5600.

```
C
C  MOLSIEV
C
 1600 CALL TRANSLT(IX,IY,IROT,IMOL,JAR)
      CALL COLOR(3)
      CALL BLOCK(IMOL(1),IMOL(2),IX,IY,IROT)
C
      CALL BLF(JAR(4),JAR(5), 1,IROT,NSCODE(1,ITYPE))
      CALL BRT(JAR(6),JAR(7),-1,IROT,NSCODE(2,ITYPE))
      CALL BUP(JAR(8),JAR(9),-1,IROT,NSCODE(3,ITYPE))
      CALL CIRCLE(2,JAR(10),JAR(11))
C
      CALL GRLBCT(IX+0,IY+10,IEN(1,ITYPE),8)
      CALL GRLBCT(IX+0,IY-10,NAME,8)
      CALL GRLBAB(IX+JMOL(1,JRO),IY+JMOL(2,JRO),'AIR',3)
      CALL GRLBAB(IX+JMOL(3,JRO),IY+JMOL(4,JRO),'AIR',3)
      CALL GRLBAB(IX+JMOL(5,JRO),IY+JMOL(6,JRO),'CO2',3)
      CALL COLOR(1)
      GO TO 9999
```

Figure A-3. Example icon graphics code (MOLSIEV icon).

An explanation for each line in figure A-2 follows:

Line 1: TRANSLT adjusts the IMOL array based on the screen coordinates (IX, IY) where the icon is to be drawn. That is, TRANSLT converts the icon coordinate data, which locates the stream lines, from being relative to the icon center to absolute coordinates relative to the lower left hand corner of the terminal screen. These new coordinates are located in the array JAR, which contains the same information in the first three locations (box height, width and remaining used array locations) as the ICOMP array. The coordinates (IX, IY) are determined by the user when locating or moving a component with a cursor pick. The variable IROT is the icon rotation angle in increments of 90˚.

Line 2: This line changes the color of the component. Color is based on the component's function (i.e., air revitalization).

Line 3: BLOCK draws the icon box (body) centered at coordinates (IX, IY) and rotated at angle IROT (must be 0˚, 90˚, 180˚, or 270˚) on the terminal screen. The height and width are IMOL(1) and IMOL(2), respectively.

Lines 4 to 6: These statements draw the icon streams. BLF draws a line to the left of the box (parallel to the x-axis) from point (JAR(4), JAR(5)) determined from the relative coordinates (IMOL(4), IMOL(5)). The parameter "1" indicates that an arrowhead is drawn to point toward the icon body. NSCODE (1, ITYPE) contains the stream code that directs this routine to draw the hit box corresponding to that stream type (see section 8.2.3 of the user's manual). BRT draws the stream line to the right (parallel to the x-axis) from (JAR(6), JAR(7)) with an arrow pointing away from the icon body (-1). BUP draws the third stream up (parallel to the y-axis) and pointing away from the point (JAR(8), JAR(9)) on the icon.

Line 7: CIRCLE draws the body hit dot, a circle with a diameter of two pixels, at the point (JAR(10), JAR(11)).

Line 8: GRLBCT draws a graphic label eight characters long centered on the point (IX, IY + 10). The label text is the component type (MOLSIEV) that is stored in IEN(i,TYPE), i=1,2. For this case, ITYPE is 16.

Line 9: In this line, GRLBCT draws the NAME (component name located in IEL(4) and IEL(5)) centered about the point (IX, IY - 10).

Line 10 to 12: These lines draw the stream labels at the coordinates in the JMOL array. The variable JRO indicates rotation angle using values of 1 to 4 for the valid rotation angles (1= 0˚, 2 = 90˚, etc.). JRO is calculated from IROT at the beginning of DRAWC. Note that JMOL is not adjusted as is IMOL. The rotation angle and absolute coordinates are included in the CALL statement (Remember that the second index of JMOL indicates the rotation angle).

Line 13: COLOR is again called to reset the color to the default.

Line 14: Control is transferred to line 9999, which dumps the buffered output and DRAWC is then exited.

1.3 Step 3: Modify "Hit" Routine

Subroutine "HIT" contains the data describing the "hit box" locations for each stream and the controller circle connection data for each component. The first modification that must be made to this routine involves the NST array (dimensioned to 65). This array contains the total number of streams associated with each component. The data are stored by equipment type. For example, the total number of streams for a PUMP component (type 27) is stored in NST(27) and has the value of 2. The total number of streams for the first additional component will be located in NST(56).

The next change inside this routine involves the component "hit box" X and Y coordinate values (refer to figure A-4). The X coordinate values are contained in the IXD array and the Y values are contained in the IYD array. Currently, both the IXD and IYD arrays are dimensioned to (65,8). The index of 65 allows for up to 10 new components to be added. The second index "8" corresponds to the maximum number of streams a component can have. The values in these arrays are input in the following manner:

The X coordinate of the hit box for stream 2 of component type 23 will be stored in the IXD(23,-1) location (refer to figure A-4).

The Y coordinate of the hit box for stream 3 of component type 36 will be stored in the IYD(36,-2) location.

To generalize: x and y coordinates for the HIT BOX of a particular component type, say ITYPE, are stored in IXD (ITYPE,i) and IYD (ITYPE,i) where i = 1-NST(ITYPE).

To add a new component, the values for IXD (52,1-8) and IYD (52,1-8) must be inserted. Zeroes are input for the locations where there are no streams (e.g., if the new component has three streams, locations IXD(52,4-8) and IYD(52,4-8) will have zero values).

```
C      DATA IXD/
C     &-15, 15,-15, 15,-70, 70, 0, 0,  0,-15, 15,-70, 70,  0, 0, 0,!EDC,SABAT
.
.
.
C     &-15, 15,  0,  0,  0,  0, 0, 0,  0,  0,  0,  0,  0,  0, 0, 0,!O2N2,BUBBLE
C     & 80,-80,  0,  0,  0,  0, 0, 0,-70, 70, 15,  0,  0,  0, 0, 0,!LIOH,MOLSIEV
C     &  0,-15, 15,-70, 70,  0, 0, 0,-15,-15, 15,-70, 70,  0, 0, 0,!SPES,SFWE
.
.
.

C      DATA IYD/
C     & 70, 70,-70,-70,-10, 10, 0, 0, 70,-70,-70,-10, 10,  0,  0, 0,!EDC,SABAT
.
.
.
C     & 70, 70,-70,  0,  0,  0, 0, 0,  0,  0,  0,  0,  0,  0,  0, 0,!O2N2,BUBBLE
C     &  0,  0,  0,  0,  0,  0, 0, 0,  0,  0, 70,  0,  0,  0,  0, 0,!LIOH,MOLSI*
C     & 70,-70,-70,  0,  0,  0, 0, 0, 70,-70,-70,  0,  0,  0,  0, 0,!SPES,SFWE
.
.
.
```

Figure A-4. Example hit box data initialization (MOLSIEV icon).

The last modification to be done to the HIT routine defines the X and Y locations of the body hit dot (circle). Each component has a two-pixel circle that lies on the outer boundary of the component icon body. This circle is used to connect the given component to a controller (CNTRLLR) or TIMER. The two arrays used for the X,Y location of this hit circle are the ICNXD and ICNYD arrays. At the present time, these are one-dimensional arrays with a maximum number of 65 values. Again, as with IXD and IYD, the data are stored according to equipment type. For example the value stored in ICNXD(16) = -27, and ICNYD(16)= -30. Therefore, the body hit dot for the MOLSIEV is located at (-27, -30) relative to the icon centroid (refer to figure A-4). Note that stream hit dots are handled automatically by the routines that draw the stream line and hit box.

1.4 Step 4: Modify "Locate" Routine

Also located in LOCATE.FOR is the array NANGLE(65), which contains the default orientations for each component (default IROT). Typically, the default angle is 0˚. If some other orientation is desired, however, it should be stored in NANGLE according to ITYPE. The MOLSIEV is NANGLE (16) = 0.

1.5 Step 5: Modify "Ssout" Routine

Subroutine SSOUT contains the stream labels for each component that appears in the .LPP printout. The character array (STRLBL) that holds these labels is currently dimensioned to 8 by 65, accommodating 65 components with a maximum of eight streams each. STRLBL is a CHARACTER*8 variable, thus, the label name can have a maximum of eight characters. For labels shorter than eight characters, the array must be padded with blanks. In the MOLSIEV example, the three stream labels are stored in STRLBL (1-3,16). Locations STRLABL(4-8, 16) are strings of 8 blanks. The DATA statement from SSOUT.FOR initializing the MOLSIEV is shown in figure A-5 (note that the statement is scarred for ten additional labels represented by the comment "DUMMY").

```
 DATA (STRLBL(i,1),i=1,8)/'AIR IN  ','H2 IN   ','AIR OUT ',  !EDC
&   'CO2 OUT ','COOL IN ','COOL OUT','        ','        '/
    ...

 DATA (STRLBL(i,16),i=1,8)/'AIR IN  ','AIR OUT ','CO2 OUT ', !MOLSIEV
&    '        ','        ','        ','        ','        '/
    ...

 DATA (STRLBL(i,65),i=1,8)/'        ','        ','        ', !DUMMY
&    '        ','        ','        ','        ','        '/
```

Figure A-5. Example hit stream label declaration (MOLSIEV icon).

1.6 Step 6: Modify "Pinit" Routine

The first modification of PINIT.FOR involves the NSCODE array. These changes are fully described in section A.1.2. It should be noted that this array is used in DRAWC only as an indicator of how to draw the hit boxes for a given stream type. In PINIT, the NSCODE array is used to set the values of the HYDRA array.

The second modification is to the array JSCODE. This array is dimensioned to JSCODE(8, 65), as is NSCODE. The first index represents the stream number and the second is for component type. The JSCODE array contains "imposed flow codes" for each of the component's streams. These flow codes are:

–1 = component stream acts like a sink
0 = no imposed flow (pass through)
1 = component stream acts like a source

As an example, the MOLSIEV has three streams: streams 1 and 2 are flow-through and stream 3 delivers CO_2 to a device located downstream. From the system level, streams 1 and 2 do not impose flow (i.e., air enters and exits because of the pressure exerted by upstream and downstream devices). Stream 3, however, appears to the system to be a source for CO_2. Thus, for the MOLSIEV component:

JSCODE (1, 16) = 0
JSCODE (2, 16) = 0
JSCODE (3, 16) = 1

Since the MOLSIEV has only three streams, the remaining values of JSCODE (i, 16) are zero.

The final modification to PINIT when adding a new component is to the array KSCODE. Like NSCODE and JSCODE, KSCODE is dimensioned to KSCODE (8, 65). This array contains the internal (to component) connection data. The possible values for KSCODE are shown below:

0 = no direct internal connection to any other component stream
1 to 8 = stream number of opposite end of internal flow path
9 = stream is connected to all other streams (i.e. SPLIT, MSPLIT)

In the MOLSIEV, air enters stream 1, passes through the component, and exits at stream 2. Therefore, stream 1 has an internal connection to stream 2. Likewise, stream 2 has an internal connection to stream 1. Stream 3 has no internal connection to another MOLSIEV stream. Thus for the MOLSIEV component:

KSCODE (1, 16) = 2
KSCODE (2, 16) = 1
KSCODE (3, 16) = 0
KSCODE (4-8, 16) = 0

1.7 Step 7: Modify “Eqsolve” Routine

The subroutine EQSOLVE has the responsibility to call the correct component requested by SOLVE based on the component type. SOLVE obtains the IEQ number from the NPCS array (see section 5.6.1). EQSOLVE is then called. From the IEQ number, the component type is determined from the IEL array; ITYPE=IEL(IEQ, 3). Based on ITYPE a computed GOTO transfers control to statement label ITYPE + 10. For example, the MOLSIEV component would cause control to go to label 26, since the MOLSIEV type is 16. EQSOLVE is scarred for 10 new components at labels 66 through 75 (ITYPE 56 through 65). At each of these statement labels, the CONTINUE statement would simply be replaced by the statement CALL NEWTYPE, where NEWTYPE would be a component subroutine having the structure discussed in appendix section 3.0.

1.8 Step 8: Update Case/A Object Library

When code is modified for any reason, the object library must be updated with the new code in order for all users to access it. The safest way to make changes to CASE/A is to copy the source routines from the CASEA$CODE directory into a user directory and make the necessary changes to the copy. When editing of the source is complete, the new routines need to be compiled to create an object file for

linking with the CASEA library. As with OPS logic, this is done by invoking the VMS FORTRAN compiler as follows:

```
$ FORTRAN/NOOPTIMIZE/DEBUG/CHECK=BOUNDS/CONT=99 filename.FOR
```

where filename.FOR is the name of the modified routine, EQSOLVE.FOR for example. If no compilation errors occur, the new object code can be linked with the rest of CASE/A with the VMS linker. For example, if adding a new component NEWCOMP, and having made the required changes to the files as described above, the LINK statement should be:

```
$LINK/DEBUG/EXE=CASEA.EXE NEWCOMP, DRAWC, HIT, SSOUT, PINIT,-
EQSOLVE, CASEA$CODE:CASEA, CASEA/LIB
```

This command causes CASEA.EXE to be created in the current default directory (this should be a user directory, not CASEA$CODE) from the object files NEWCOMP, DRAWC, HIT, SSOUT, PINIT, and EQSOLVE in the user directory; CASEA.OBJ and CASEA.OLB (CASE/A object library) in CASEA$CODE. The linker will use the object code listed first in the link statement, thus abandoning the old subroutines located in CASEA$CODE:CASEA.OLB (these are not removed, just not linked). Thus, the library remains intact and costly errors can be avoided.

Since this process is likely to be required several times during the construction of a new component, it may be wise to copy CASEA$CODE:LINKCAS.COM into the user directory and modify the LINK statements contained therein in the above manner. It would then be a simple matter to relink by the following statement:

`$ @LINKCAS` or if OPS Logic were involved: `$ @LINKCAS OPSFILE`

When modifications are complete and thoroughly tested, the CASE/A library is ready to be updated. It is extremely important to test code modifications rigorously, as the CASE/A library is public and all users rely upon its integrity. It is a safe practice to retain old versions of code in case errors should occur when changes are made.

To update the library, copy the new source routines into the CASEA$CODE directory, retaining the previous versions (default version limits should be greater that 1 for CASEA users). Next, change default directories to CASEA$CODE:

```
$SET DEFAULT CASEA$CODE
```

The command procedure CD.COM is in this directory and is used to compile the source and replace the objects in the library CASEA.OLB. This is done by entering the command:

```
$ @CD FILE
```

at the DCL prompt. “FILE” is one of the new source files and should not include the extension (i.e., EQSOLVE not EQSOLVE.FOR). When this is done for each new source file, all users should relink using LINKCAS (see section 2.1 of the user’s manual).

The final step is to update the library list so that in the event the library needs to be recreated the new component routine will be updated as well. The files LIB1LIST.COM and LIB4LIST.COM contain the names of each CASE/A subroutine source file. The new component routine filename (NEWTYPE) should be added to the end of LIB4LIST.COM. The last two lines are shown below in figure A-6.

```
ELIMINAT,       EXCHAN,         FIT,                    GRLBAB,     -
INTERSEC,       PLTOVR,         POINT,                  RANK,       -
SCAN,           SCREDT,         SELECT,                 SIGMA,      -
SORT,           TDPLOT,         UNION
```

Figure A-6. Modified library list (LIB4LIST.COM).

See appendix C of the user's manual for creating a library using LIBCREATE.

2.0 Data Base Construction

2.1 Data Base File Description

When a new component is created, a new data base must be generated to support that component. A listing of the files in the data base directories reveals that a data base exists for every component except for BUBBLE. Also, each data base consists of at least eight files:

```
NAME.DDF        NAME.STT*
NAME.DAT        NAME.USR*
NAME.ED1        NAME.SUB*
NAME.ED2
NAME.SCR        * - these files are maintenance files automatically created by
                the data management system and their function is unimportant to
                the CASE/A programmer and beyond the scope of this manual.
```

2.1.1 Data Definition File

The data definition file (DDF) specifies how the component data (stored in binary form) are interpreted. A generic DDF is shown in figure A-7. The first 12 entries in every component DDF store the same type of information. The remaining items are user-specified and define the inputs, outputs, and benchmarks associated with the component. In the generic DDF, words 16 to 73 are inputs, words 67 to 70 are outputs, and words 116 to 175 are benchmarks. The labels shown (INPUT CONSTANT #1, etc.) should be modified to indicate the particular parameter of interest (32 characters maximum). The header line and the three-digit integers preceding each item are described in section 3.4.

2.1.2 Binary Data File

This file (.DAT) contains the actual component data. The data are stored in binary form (note that this file cannot be printed in the typical ASCII form). The NEWCOMP.DAT file is automatically updated as components of NEWCOMP type are used in model development. It should be noted that the data associated with every NEWCOMP component in EVERY model at a given installation site are stored in this file. If NEWCOMP.DAT is deleted, all NEWCOMP data will be lost. This is true for any COMPONENT.DAT file.

2.1.3 Full Screen Editor Templates

The .ED1 and .ED2 files store information enabling data from the .DAT file to be read (according to the .DDF) and presented in a full screen format. The .ED2 file is a text file that contains all of the parameter labels for a component. These template files are located in the directory with the .DDF and .DAT files ([CASE.DATA]). This directory is specified by the VMS logical name CASEA$DATA. The .ED1 file is created by running TDMS and executing the SCREEN command (from within the [CASEA.DATA] directory. The details of this command may be obtained via the online help utility.

```
81 78 77 15000 78  0 0
002 002 001 CASE NAME
002 002 003 SUBSYSTEM NAME
002 002 005 COMPONENT NAME
003 001 007 NUMBER OF INPUTS
003 001 008 NUMBER OF OUTPUTS
003 001 009 NUMBER OF BENCHMARKS
003 001 010 CDEL LOCATION
003 001 011 TDEL LOCATION
003 001 012 PDEL LOCATION
003 001 013 POWER  ,WATTS
003 001 014 WEIGHT ,LBM
003 001 015 VOLUME ,CUBIC FEET
003 001 016 BED FLAG (1-5A,2-EMRC)
003 001 017 ADSORBENT BED WEIGHT       ,LBM
003 001 018 HEAT OF ADSORPTION ,BTU/LBM-CO2
003 001 019 BAKEOUT TEMP SET POINT   ,DEG-F
003 001 020 BED LOAD CAPACITY INCREASE FACT
003 001 021 OPERATIONAL FLAG(0-ABS, 1-DES)
003 001 022 BED SHELL ENV CNVCTN G,BTU/HR/F
...
003 001 066 **OPEN INPUT**
003 001 067 MAXIMUM CO2 BED LOADING, %
003 001 068 **OPEN INPUT**
003 001 069 TOTAL CO2 REMOVED         , LBM
003 001 070 FINAL CO2 BED LOADING     , LBM
003 001 071 AIR IN MAX, LBM/HR
003 001 072 AIR IN MIN, LBM/HR
003 001 073 AIR IN NOM, LBM/HR
003 001 074 AIR OUT MAX, LBM/HR
003 001 075 AIR OUT MIN, LBM/HR
003 001 076 AIR OUT NOM, LBM/HR
003 001 077 CO2 OUT MAX, LBM/HR
003 001 078 CO2 OUT MIN, LBM/HR
003 001 079 CO2 OUT NOM, LBM/HR
001 001 080 MOD DATE
002 001 081 SECURITY
$$EX
```

Figure A-7. Example data definition file (DDF).

2.1.4 Script File

The script file (.SCR) is a text file that is automatically created for every data base. It serves as a macro file that is executed each time the data base is loaded. This file is useful for output data bases but should not be used for component data bases.

2.2 Adding/Modifying Component Data Base Files

Adding or modifying the component data base files requires the use of a text editor (such as EDT on the VAX/VMS) and the TDMS data management tool. The best source for information regarding TDMS can be obtained from executing TDMS and reviewing the help screens. The following is a brief list of steps required to add/modify a component data base:

Step 1: If modifying an existing component database, make a backup copy of the original.

Step 2: Update or create the .DDF file using a text editor (such as EDT on VAX/VMS). Refer to TDMS Data Structures Help menu or section 3.4 of this manual for the formatting conventions.

Step 3: (a) If adding a new component, the .DAT file will be updated as new components are added under the CASE/A model construction process. To add default settings for the new component, execute TDMS and use the ENTRY command.

(b) If modifying an existing component, execute the program INSERT as follows:

```
RUN/NODEBUG CASEA$DATA:INSERT
```

This program will prompt for changes to the .DAT file and insert 0.0 into all records for the new fields in that component database.

Step 4: Update/Add the .ED2 file using the text editor (e.g. EDT). This screen layout can be changed as necessary to add or move fields that will be displayed from this component's database. In addition the .ED2 file must be consistent with the .ED1 file in order for the screen display for the component to work correctly. Note the row/column positions of the new fields added to the .ED2 file, so that these can be added to the .ED1 file.

Step 5: (a) If adding a new component, use TDMS command "SCREEN" to position items in the .DDF file. This will correspondingly update the .ED1 file as items in the .DDF file are positioned.

(b) If modifying an existing component, use the VAX/VMS editor to update the .ED1 file. The format is described in the TDMS help panel for "SCREEN."

3.0 Component Fortran Routine

The most important, and perhaps the most difficult, task in creating a new component is creating the FORTRAN code to model a component. All of the components are unique in nature due to the architecture of the CASE/A command processor. Each of the subroutines are broken into four discrete sections, depending upon the state of execution of the solution routine. These sections pertain to the following aspects of a systems-level simulation:

Subroutine Block	Execution State
100 Block	Called one time at the beginning of the simulation.
200 Block	Called every component iteration. Called a maximum of NLOOP times.
300 Block	Called after each successful time step convergence or after a maximum number of loops have been exceeded.
400 Block	Called one time at the end of a simulation.

The general types of operations that are executed in each of these blocks will be discussed in the following sections. The final section provides a step-by-step description for one of the existing CASE/A library subroutines. This subroutine is for an ideal heater and is called HEATER. This is one of the simplest component models in the CASE/A system and will allow the user to get a feel for the generalized flow of data into and out of the component.

3.1 Data Initialization Segment (100 Block)

The data initialization segment portion of CASE/A occurs only one time for each component within a given simulation. The first statement within the 100 block (for components with data files) is the command CALL CONINIT. This command loads all of the data for that particular component type from mass storage into random access memory. Other types of operations performed in this block include initializing the CON array outlet locations to zero, initializing variables that stay constant throughout the entire simulation, setting the plot array (D array) locations to zero, and setting the appropriate operation flags to their initial values.

3.2 Iterative Solution Segment (200 Block)

The iterative solution segment performs the operations associated with the simulation of that component type's physical behavior. The mass flow, thermal, and hydraulic calculations are performed according to the operating characteristics for each specific component within a given case. The general types of operations executed in the 200 block also depend on the type of component, active or passive. Active components will generate flow according to a rate specified by the user or a characteristic flow curve. Passive devices accept flow from the upstream component and operate on the control volume appropriately depending on the temperature, pressure, and composition of the inlet stream.

The iterative segment of a component becomes more complicated if the component in question requires a transient mass, thermal, or combined response. Transient response routines require the user to save the initial conditions from the last successful time step convergence into temporary values, and also update the transient values from iteration to iteration within the component. Often times, the transient routine will require a smaller time step than the system time step, hence the routine must be "fooled" by iterating within that component's 200 block for a number of loops given by the following relation:

Number of iterations inside the 200 block for a routine whose internal time step is calculated to be 5 seconds and is inside a system whose time step is 60 seconds:

# of Loops = (60 S/5 S) = 20

Some of the more detailed transient routines currently available within the CASE/A library are MOLSIEV, DEFLOW, and DEHUM and SABAT.

Specific requirements must be met in the 200 block of each component routine. The component pressures, mass flow rates, mass fractions, and specific heats must be passed from the inlet to the outlet so that the downstream component will have values to operate on if an error condition is reached. This is usually done near the beginning of the block . The next portion of the 200 block contains the inputs that are read from the CON array. These values are likely to change with time, hence the user can change these through operations logic. This section varies in size depending on the complexity of the process at hand. The actual "iterative" portion of the 200 block is usually based on a mass or temperature relationship. In other words, the iterative portion of the block is executed until some user input relaxation involving a temperature or constituent mass fraction is met. This section also varies greatly in size depending on the nature of the process. The final portion of the 200 block contains operations that are similar for all components. The cabin heat load and conductor (CLOAD and GSUM) arrays are updated based on the final component temperature, outlet stream mass fractions and flow rates are set, and the stream outlet pressures are calculated depending upon the user's preference for calculation type (see section 8 of the user's and programmer's manual).

3.3 Post-time Step Wrap-Up Segment (300 Block)

The post-time step wrap-up segment of each component routine is called one time per successful system time step (at the end of the time step). The CON array output locations are updated at this time along with any benchmark parameters that tracked on a per time step basis. If the component in question has an internal accumulation stream, it must be updated within the 300 block also. The 300 block varies in size depending on the number of output and benchmark locations that are reserved by the new component.

3.4 Postsimulation Wrap-Up Segment (400 Block)

The post simulation wrap-up segment of the code is executed one time per simulation (at the simulation end). The structure of the 400 block is very similar for every component. The data updated in

this section include the benchmark parameters. Note that the call to the routine EQWRIT is no longer used in the current version.

3.5 Example Component Subroutine – Heater

Figure A-8 shows example source code for an ideal heater. It is documented so that the general logic of a component subroutine can be determined. The programmer is encouraged to investigate the other component subroutines in CASEA$CODE to obtain a better understanding of each model.

```
MINS2>type heater.for /page
C *** DEC/CMS REPLACEMENT HISTORY, Element HEATER.FOR
C *** *2     2-OCT-1990 15:34:41 ANDERDE "this is the cosmic (version
                                         4.1) version"
C *** *1     6-SEP-1990 08:58:55 ANDERDE "Initial element creation from
                                         user4$:[ed62.casea4.code]"
C *** DEC/CMS REPLACEMENT HISTORY, Element HEATER.FOR
      SUBROUTINE HEATER
C
C  COMPCOM.INC CONTAINS COMMON BLOCKS OF ALL CASE/A ARRAYS
C  NECESSARY FOR COMPONENT OPERATIONS.
C
      INCLUDE 'compcom.inc'
      INCLUDE 'com_io.inc'
C
C  COMPUTED GOTO DEPENDING ON SOLUTION SEGMENT
C
      GO TO (100,200,300,400) MFLAG
C*********************************************************************
C
C  INITIALIZE "CON" ARRAY BENCHMARK LOCATIONS
C
  100 CALL CONINIT
C
C     RESERVE D(IEQ,11) AS FLAG THAT INDICATES
C     IF HEATER IS ON (0) OR OFF (1.0) FOR
C     CUMMULATIVE OPERATING TIME CALCULATION
C     PERFORMED IN "300" BLOCK
C
      D(IEQ,11)   = 0
      CON(NOUT+2) = 0   ! INITIALIZE OUTPUT
      GOTO 499
C*********************************************************************
C
C  COPY UPDATES INLET STREAMS' PROPERTIES AND FLOW RATES, AND PREPARES
C  "ACTIVE SET" FOR CONVERGENCE CHECKS IN SUBROUTINE KHECK.
C
  200 CALL COPY(2)
C
C  PASSIVE CHECKS FOR PULL THROUGH FLOW IN COMPONENTS WITH PASSIVE
C  INLET/OUTLET STREAMS
C
      CALL PASSIVE(2,1,2)
      D(IEQ,11) = 0
      IF (CFLAG) GOTO 499
```

Figure A-8. Example source code for an ideal heater component.

```
C
C
C  PASS FLOW RATE AND COMPOSITIONS FROM INLET TO OUTLET AS A SAFETY
MEASURE.
C
      DO 210 I = 1,NTRACK
  210 C(I,npt2) = ABS(C(I,npt1))
C
C
C  PASS TEMPERATURE TO STREAM 2
C
       PRO(2,IEQ,2) = PRO(2,IEQ,1)
C
C  DEFINE PERTINENT LOCAL VARIABLES
C
      FLOW  = ABS(C(1,npt1))
      CP    = PRO(3,IEQ,1)
      TIN   = PRO(2,IEQ,1)
      TNEED = CON(NINP+1)
C
      IF (TIN .GT. TNEED) THEN
       D(IEQ,11) = 1.0
       QNEED = 0
       TNEED = TIN
      ELSE
       QNEED = FLOW * CP * (TNEED-TIN)
      ENDIF
C
      CON(NOUT+1)  = QNEED
C
C  SET OUTLET TEMPERATURE
C
      PRO(2,IEQ,2) = TNEED
C
  298 CONTINUE
C
C  THIS ROUTINE CALCULATES THE PRESSURE DROP THROUGH THE INTERNAL
C  CONNECTIONS OF A PASSIVE, FLOW THROUGH DEVICE BASED ON A USER SUPPLIED
C  EQUIVALENT LENGTH AND DIAMETER (NOTE THAT DIA MUST BE CONVERTED TO
FEET).
C
      CALL PIPEDP(2,1,2,CON(NINP+2),CON(NINP+3)/12.0)
C
C  CALL ROUTINE TO CHECK CONVERGE AT A SYSTEM LEVEL (FROM THE ACTIVE SET
C  ESTABLISHED IN Subroutine COPY).
C
  299 CALL KHECK(2)
      GOTO 499
C*********************************************************************
C
C  SAVE DESIRED VALUES FOR BENCHMARK CALCULATIONS
C  NOTE THAT BENCHMARKS WORK IN “THREES”, I.E., MAX,MIN, AND NOM.
C  THE ROUTINE BENCH ASSUMES THAT IF BENCHMARK 1 IS SPECIFIED, THEN
C  BENCHMARKS 2 AND 3 MUST ALSO BE UPDATED.
C
```

Figure A-8. Example source code for an ideal heater component (continued).

```
  300 CALL BENCH(1,ABS(C(1,npt1)))
      CALL BENCH(4,PRO(2,IEQ,1))
      CALL BENCH(7,CON(NOUT+1))
      IF (D(IEQ,11) .EQ. 0) THEN
       CON(NOUT+2) = CON(NOUT+2) + STEP
      ELSE
       write(iuo,*)
       CALL TEK_ADV(1)
       write(iuo,*)'HEATER WARNING: INLET TEMP > DESIRED OUTLET TEMP.'
       CALL TEK_ADV(1)
       WRITE (6,310) (IEL(IEQ,K),K=4,5),IEQ
  310  FORMAT(1X,'EQUIP NAME: ',2A4,'    EQUIP IEQ: ',I5)
       CALL TEK_ADV(1)
       write(iuo,*)'NO HEATER POWER APPLIED: CONTINUING SIMULATION...'
       CALL TEK_ADV(1)
      ENDIF
      GOTO 499
C*********************************************************************
C
C  CALCULATE NOMINAL VALUES OF BENCHMARKS (3RD VALUE).
C
  400 IF (TIME .NE. STRT) THEN
       CON(NBEN+3) = CON(NBEN+3) / (TIME-STRT)
       CON(NBEN+6) = CON(NBEN+6) / (TIME-STRT)
      ENDIF
C
      IF (CON(NOUT+2) .NE. 0) THEN
       CON(NBEN+9) = CON(NBEN+9) / CON(NOUT+2)
      ENDIF
C
C
C  NULL CALL, THIS ROUTINE IS NO LONGER USED.
C
      CALL EQWRIT
  499 RETURN
      END
C
```

Figure A-8. Example source code for an ideal heater component (continued).

APPENDIX B. GLOSSARY OF LABELED COMMON BLOCK VARIABLES

The following table provides the definition of all major variables that are in common blocks. It describes the function of the variable and what is represented by each index. These common blocks are located in the following "INCLUDE" files:

COMPCOM.INC denoted as "C"
FRAMECOM.INC denoted as "F"
GRAPHCOM.INC denoted as "G"
COM_IO.INC denoted as "I"
SOLVCOM.INC denoted as "S"
UTILCOM.INC denoted as "U"

Also "P" denotes common block used for plot functions that is declated directly in subroutines where referenced.

These files are located in the CASEA$CODE directory.

VARIABLE	DATA TYPE	COMMON BLOCK NAME	INCLUDE FILE	DESCRIPTION
AFLNAM	CHAR*80	/ARCHIVE/	G	Contains archive file name for model to be archived. Used in the ARCHIVE routine.
ARCH_DIR	CHAR*40	/IO_UNITS/	I	Contains directory name for the directory containing archive files. Used in the ARCHIVE routine.
ARCHSET_NAME	CHAR*8	/ARCHIVE/	G	Contains archive set name for model to be archived. Used in the ARCHIVE routine.
C(54,2000)	REAL*4	/CPRO/	C,G,S,U	Stream constituent array. See section 7.2.2.
CFLAG	LOGICAL*4	/CON/	C,G,S,U	When CFLAG is TRUE the 200 block of components whose input/output arrays have not changed since the last iteration is skipped.
CLOAD(25,3)	REAL*4	/CABIN/	C,U	Sum of component heat transfer from components in an assigned subsystem. The first index is the relative cabin number. The second is for convection, radiation, and conduction respectively.
CON(75000)	REAL*4	/CON/	C,G,S,U	Component data array. See section 7.2.1.
CONFLAG	REAL*4	/CON/	C,G,S,U	CONFLAG is used by the controller for various "ACTIVATION CODES" CONFLAG can be between 0 and 9. See CNTRLLR component in chapter 10 of the user's manual.
CRAD(25)	REAL*4	/CABIN/	C,U	Effective radiation temperature in cabin environment.
CSPLIT(250,9)	REAL*4	/MSPLIT/	G	The first index of CSPLIT is the relative number of the split component and the second is 1=number of split legs, 2-9 = split fraction for each leg.
CSTR(25)	REAL*4	/CABIN/	C,U	Structure temperature.
CTEMP(25)	REAL*4	/CABIN/	C,U	Internal cabin temperature.

D(1000,50)	REAL*4	/PLOT/	C,G	D gives 50 internal storage location for 1000 components. This is internal and unique to each component. The 50 locations should be initialized in the components 100 block. This is so that when changes are made or new components added, the common.inc files will not need changing.
DEVICE	CHARACTER*12	/DEV/	C,G,S,U	VMS logical disk name (i.e. DISK$2).
DIRECTORY	CHARACTER*20	/DEV/	C,G,S,U	VMS directory (i.e. ECLS.CASEA).
DIRNAME	CHARACTER*80	/DEV/	C,G,S,U	Concatenation of the above to form a complete directory path (i.e. HSV::DISK$2:[ECLS.CODE]).
END	REAL*4	/EXQ/	C,F,G,S,U	Simulation termination time in hours.
ERRFLAG(1000,3)	LOGICAL*4	/ERRFLG/	C	A conditional flag used by the H2OSEP component.
FLOCOND(2000,5)	REAL*4	/PRESSURE/	C,G,S,U	This array contains the conductor data for each flow conductor. The first index is the conductor number, the second is: Node A number, Node B number, mass flow rate, pressure drop, and C array pointer for values of 1, 2, 3, 4, and 5.
GSUM(25,3)	REAL*4	/CABIN/	C,U	Sum of conductance values from assigned subsystem components to assigned cabin. First index is relative cabin number, second index is convection, radiation, and conduction, respectively.
HOST	CHARACTER*32	/DEV/	C,G,S,U	VMS network host name (i.e. HSV).
HYDRA(2,1000,8)	REAL*4	/CPRO/	C,G,S,U	The Hydra array is used to determine the pressure solution code for a given stream. The code indicated hydraulic characteristics of the stream. The first index indicates the hydraulic code number. The second indicates the relative component location. The third is the stream number of that component.
IABORT	INTEGER*4	/ARCHIVE/	G	Flag used to indicate selection by user to exit routine. Used in RETRIEVE, MERGE_IN and MERGE_OUT.
IARCHIVE	INTEGER*4	/ARCHIVE/	G	Switch set to indicate to MERGE_OUT that ARCHIVE routine is calling it.
IAUTOPLOT	INTEGER*4	/ARCHIVE/	G	Set to indicate ploting is complete. Used in AUTOPLOT and OPENDB_X.
IBBLAB(25,8)	INTEGER*4	/BBOX/	G,S	Black box label array. The first index is the relative black box number. The second index is a 4 character label for each of the eight possible streams.

ICL(2000,20)	INTEGER*4	/CONSUME /	C,F,G,S,U	ICL is the connection data array. The first index is the connection number. The second index is as follows: INDEX DATA 1 IEQ of "A" side component 2 Stream # of "A" side component 3 IEQ of "B" side component 4 Stream # of "B" side component 5 Unused 6 Number of segments 7-16 (x,y) coordinates of midpoints 17 Equivalent length 18 Equivalent diameter 19-20 Unused
ICNXD(65)	INTEGER*4	/STRM/	G,S	X location of body hit circle relative to icon center. Index is component TYPE number.
ICNYD(65)	INTEGER*4	/STRM/	G,S	Y location of body hot circle relative to icon center. Index is component TYPE number.
ICP(1000)	INTEGER*4	/POINT/	C,S,U	Contains the pointer to the start of each segment of the CON array. (ICP(IEQ)). See Section 7.2.1.
ICS(2)	INTEGER*4	/CONSUME /	C,F,G,S,U	Current casename. ICS(1) is the first 4 characters, ICS(2) is the second 4 characters.
IEL(1000,15)	INTEGER*4	/IEL/	C,F,G,S,U	IEL is used to store system and graphic data for each component. The first index is the relative equipment number (IEQ), the second is : INDEX DATA 1-2 subsystem name 3 component type 4-5 component name 6 x location on subsystem screen 7 y location on subsystem screen 8-11 unused 12 cabin assignment data 13 record number in data base 14-15 unused
IEN(2,65)	INTEGER*4	/CONSUME /	C,F,G,S,U	This is the list of CASE/A component names. For values of the first index of 1 and 2 IEN holds the first 4 or last 4 characters of the name. The second index is the component TYPE (ITYPE) as discussed in chapter 10.
IEQ	INTEGER*4	/EXQ/	C,F,G,S,U	The relative equipment number of the component currently being considered. IEQ is advanced in SOLVE according to the PCS.
IFRAMEFLAG	INTEGER*4	/KSCREEN_ LOC/	C,F,G,S,U	Used by TEK_ADV to determine wether to erase the screen.

ILB(150,10)	INTEGER*4	/CONSUME /	C,F,G,S,U	LABEL data array. The first index is the label number. The second is: INDEX DATA 1-2 subsystem name 3 stream number 4 location in target array 5-6 label text 7 target array 1=c,2=pro,3=con 8 x location 9 y location 10 IEQ number See the LABELS command in the user's manual.
INOTE(26,150)	INTEGER*4	/CONSUME /	C,F,G,S,U	NOTE data array. The first index contains: INDEX DATA 1-2 subsystem name 3 justification code 4 text size 5-24 text 25 x-location 26 y-location The second index is the note number.
IPCS(2000,2)	INTEGER*4	/IEL/	C,F,G,S,U	Used to store PCS data.
IPLT(20,300,4)	INTEGER*4	/PLTSET/	P	Plot item code array. First index is plotset number NSET, second is item number (LITEM), the last corresponds to IEQ, Array code, location in array, stream number. The array codes are 1=CON, 2=C, 3=PRO, 4=USERCON, 5=D.
IPREC(20)	INTEGER*4	/PLTSET/	P	Record number of the plotset in the parent data base.
IPSNAME(20,2)	INTEGER*4	/PLTSET/	P	PLOTSET name. First index is plotset number (NSET) second is name stored in two four-byte integers.
IPSTIT(300,8)	INTEGER*4	/PLTSET/	P	Title associated with item to be plotted. Title is eight four-byte integers for a total of 32 characters.
IPTFLG	INTEGER*4	/EXQ/	C,F,G,S,U	If flag is set false, then pressure and temperature convergence will not be considered.
ISS(2)	INTEGER*4	/CONSUME /	C,F,G,S,U	Subsystem name for the current subsystem ISS(1) is the first four characters, ISS(2) is the second four characters.
IUI	INTEGER*4	/IO_UNITS/	I	Contains unit number from which input is read. Used throughout CASE/A code.
IUNIT	INTEGER*4	/IO_UNITS/	I	Contains unit number for additional input device.
IUNLOCK	INTEGER*4	/IO_UNITS/	I	Contains value designating that a file is unlocked. 1- locked, 0-unlocked.
IUO	INTEGER*4	/IO_UNITS/	I	Contains unit number to which output is written. Used throughout CASE/A code.

IXD(8,65)	INTEGER*4	/STRM/	G,S	X location of hit box relative to icon center. First index is stream number , second is component TYPE number.
IYD(8,65)	INTEGER*4	/STRM/	G,S	Y location of hit box relative to icon center First index is stream number , second is component TYPE number.
JCL(2000,4)	INTEGER*4	/IEL/	C,F,G,S,U	JCL is the same as ICL except the second index contains only IEQ and stream numbers in the same order as ICL.
JCON	INTEGER*4	/POINT/	C,S,U	A pointer to the con array used during loading of data.
KOUNT(20)	INTEGER*4	/PLTSET/	P	Number of items to be plotted in each plotset (300 max).
KSCREEN_XINC	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Number of pixels to move the cursor.
KSCREEN_XLOC	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Current X location of the cursor on a Tektronix™ terminal in pixels from the lower left corner.
KSCREEN_XMAX	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Width of the screen.
KSCREEN_XMIN	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Left edge of the screen.
KSCREEN_YINC	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Number of pixels to move the cursor.
KSCREEN_YLOC	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Current Y location of the cursor on a Tektronix™ terminal in pixels from the lower left corner.
KSCREEN_YMAX	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Height of the screen.
KSCREEN_YMIN	INTEGER*4	/KSCREEN_LOC/	C,F,G,S,U	Bottom of the screen.
LBEN	INTEGER*4	/POINT/	C,S,U	Number of benchmark data (see section 7.2.1).
LCPU(4)	INTEGER*4	/IEL/	C,F,G,S,U	LPCU contains the record number for the labels, control, plot, and usercon data bases in the Master data base.
LFXD	INTEGER*4	/POINT/	C,S,U	Length of fixed data. Always 9 (see section 7.2.1).
LINP	INTEGER*4	/POINT/	C,S,U	Number of input data (see section 7.2.1).
LITEM	INTEGER*4	/PLTSET/	*	Item to be plotted for a particular PLOTSET.
LOUT	INTEGER*4	/POINT/	C,S,U	Number of output data (see section 7.2.1).
LPCS	INTEGER*4	/EXQ/	S,G,F,C,U	Used with the map data to step through the PCS.
LTITLE(150)	INTEGER*4	/LABELS/	C,S,U	The constituent labels contained in the LABELS data base are stored in this array. Each label occupies three elements of this array. For example CO2 is: LTITLE(10)='CARB' LTITLE(11)='ON D' LTITLE(12)='IOX '

MACTFLAG(20)	INTEGER*4	/PLTSET/	*	Stores a number for the relative PLOTSET which equates to the OPS Logic segment during which that set is active. For example MACTFLAG(1) = 0 indicates relative plotset number 1 is inactive. MACTFLAG(3) = 4 indicates PLOTSET 4 is active during OPS4. OPS4 is the default.
MATRXFLG	INTEGER*4	/PRESSURE/	C,G,S,U	Indicates wether matrix solution or feedback solution is desired.
MFLAG	INTEGER*4	/EXQ/	C,F,G,S,U	Flag to determine code block of component routine to execute.
NACT	INTEGER*4	/SORT/	C,G,S,U	The number of active components in a case. (All components minus bubbles).
NBEN	INTEGER*4	/POINT/	C,S,U	Con array pointer to benchmark data (see section 7.2.1).
NBOX(25,3)	INTEGER*4	/BBOX/	G,S	The first index is the relative black box number. The second index determines the number of input streams, number of output streams, and relative component number for index values of 1, 2, and 3, respectively.
NCAB	INTEGER*4	/CABIN/	C,U	Relative cabin number.
NCOMP	INTEGER*4	/EXQ/	C,F,G,S,U	Number of component types.
NCON	INTEGER*4	/EXQ/	S,G,F,C,U	Number of connection streams in a case.
NEQ	INTEGER*4	/EXQ/	C,F,G,S,U	Number of components in the case. Not to be confused with NACT.
NFXD	INTEGER*4	/POINT/	C,S,U	Con array pointer to first data location (see section 7.2.1).
NINP	INTEGER*4	/POINT/	C,S,U	Con array pointer to input data (see section 7.2.1).
NLAB	INTEGER*4	/CONSUME/	C,F,G,S,U	Total number of LABELS for the case.
NLPCS(2000)	INTEGER*4	/SORT/	C,G,S,U	Stores the values of LPCS by component number (IEQ).
NNOTE	INTEGER*4	/CONSUME/	C,F,G,S,U	Total number of notes in a case.
NODEINFO(1000,3)	INTEGER*4	/PRESSURE/	C,G,S,U	Used for the pressure drop solution, the first index contains the node number for a 1000 nodes, the second contains relative equipment number, stream number, and C array pointer (NCPT) for values of 1, 2, and 3, respectively.
NODENO(3,1000,8)	INTEGER*4	/PRESSURE/	C,G,S,U	Used for the pressure drop solution, NODENO(i,IEQ,ISTR) contains the node number, imposed flow code, and C pointer (NCPT) for i=1,2 and 3. for each equipment (IEQ) and stream(ISTR).
NOUT	INTEGER*4	/POINT/	C,S,U	Con array pointer to output data (see section 7.2.1)
NPCONDS	INTEGER*4	/PRESSURE/	C,G,S,U	Number of conductors in the solution.
NPNODES	INTEGER*4	/PRESSURE/	C,G,S,U	Number of pressure nodes in the solution.

NPT(8)	REAL*4	/CPRO/	C,G,S,U	NPT(ISTR) points to one of the 1500 C array locations for a particular stream number (ISTR) of a component.
NROT(1000)	INTEGER*4	/ROTATE/	G	Rotation angle of component from default in degrees. Must be in increments of 90˚.
NSEQ(2000)	INTEGER*4	/SORT/	C,G,S,U	Component IEQ numbers in the order of the PCS.
NSET	INTEGER*4	/PLTSET/	*	Relative PLOTSET number.
NSORT(2000)	INTEGER*4	/SORT/	C,G,S,U	Used in SORTIEQ to sort the relative equipment numbers and determine the pseudo compute sequence.
NSSCAB(5,100)	INTEGER*4	/ASSIGN/	F,G,S,U	The first index contains the subsystem name for i=i,2; cabin name for i=3,4 and the cabin relative component number (relative to other assigned cabins) for i=5. The second index is for up to 100 assigned subsystems.
NST(65)	INTEGER*4	/STRM/	G,S	Number of “external” streams for each component. Index is component TYPE number.
NSYSLOOP	INTEGER*4	/EXQ/	C,F,G,S,U	Number of iterations at the current time step.
NTRACK	INTEGER*4	/EXQ/	C,F,G,S,U	Number of constituents tracked.
NUM_ARCHIVED_PLOTSET	INTEGER*4	/ARCHIVE/	G	Number of plotsets stored in the current archive file selected. Used in ARCHIVE, MERGE_IN, MERGE_OUT, and RETRIEVE.
NUMBOX	INTEGER*4	/BBOX/	G,S	Number of black boxes (25 maximum).
NUMCAB	INTEGER*4	/ASSIGN/	F,G,S,U	Number of cabins to which subsystems have been assigned.
NUMSP	INTEGER*4	/MSPLIT/	G	The total number of MSPLTS.
NUMSS	INTEGER*4	/ASSIGN/	F,G,S,U	Number of subsystems assigned to a cabin.
NUMTYPE(65)	INTEGER*4	/IEL/	C,F,G,S,U	Contains the number of each type of equipment NUMTYPE(ITYPE).
PINITIAL	REAL*4	/PRESSURE/	C,G,S,U	Initial guess at pressures which are undefined by the user.
PLOTSET_ARCHIVED	CHAR*8	/ARCHIVE/	G	Contains name of plotset in archive. Used in ARCHIVE, MERGE_IN, MERGE_OUT, RETRIEVE.
PRO(12,100,8)	REAL*4	/CPRO/	C,G,S,U	Stream properties array (see section 7.2.3).
RDPLUS	LOGICAL*4	/GRAPHLOG/	U	When RDPLUS is false, the ReDraw command does not write the labels to the screen. RDPLUS is true if the command is RD+, otherwise it is false.
RELAXX(1000,8,4)	REAL*4	/CON/	C,G,U,S	Storage for the relaxation values at the current iteration. Referenced as RELAXX(IEQ,NSTR,I) where IEQ is the relative equipment number; NSTR is the stream for that component. For values of I = 1, 2 ,3 and 4 respectively, I is relaxation value for mass flow rate (total or constituent), temperature, pressure, and the constituent number which had highest relaxation value for flow rate.

RELX	REAL*4	/EXQ/	C,F,G,S,U	The relaxation criteria which must be achieved prior to time incrementation.
RXCC	REAL*4	/EXQ/	C,F,G,S,U	Maximum change in percent change in relaxation values for all streams.
STEP	REAL*4	/EXQ/	C,F,G,S,U	Time step interval in hours.
STRT	REAL*4	/EXQ/	C,F,G,S,U	Simulation start time in hours.
SYSDAMP	REAL*4	/EXQ/	C,F,G,S,U	A Damping factor used by some hydraulic routines.
TDMS_FLAG	LOGICAL*4	/TDMS_FLAGS/	G	A flag to tell when IPU commands are in effect. This flag toggles some commands so that they only work when it is set. Others, such as solve, do not work when it is TRUE. TDMS_FLG is set to TRUE when the IPU command is issued.
TERMTYPE	REAL*4	/DEV/	C,G,S,U	Unused
TIME	REAL*4	/EXQ/	C,F,G,S,U	Current time in hours.
USERCON(100)	REAL*4	/USERCON/	C,G,S,U	An array for storing data specific to a user's model (see section 7.2.4).

In addition to the above, the file CASE$CODE:SMMARY.INC contains the following variables which are output as results in the .LPP file. The SUMMARY common block is used to develop the consumables summary data at the completion of a simulation.

```
  COMMON/SUMMRY/CREW1(25),XMETAO2(25),XMETACO2(25),DRINK1(25),
     &  URIN1(25),
     &  XLEAKN2(25),XLEAKO2(25),O2GENH2O(25),O2GENH2(25),O2GENO2(25),
     &  EDCH2(25),CO2REMO2(25),EDCO2(25),CO2REMH2(25),CO2REM(25),
     &  CO2RMWAT(25),CO2RMH2O(25),CO2REDH2(25),CO2RDH2O(25),
     &  CO2RDCH4(25),CO2RED(25),BOSCHH2(25),CO2REDC(25),WASHH2O(25),
     &  WASHLOAD(25),FLUSH1(25),URINMIX(25),COND(25),STMASS(1000),
     &  XMASS1(1000),DMASS(1000)

CREW1          Number of crew members
XMETAO2 Metabolic O2 requirement
XMETACO2       Metabolic CO2 produced
DRINK1         Metabolic Water requirement
URIN1          Urine Production
XLEAKN2 N2 leakage
XLEAKO2 O2 leakage
O2GENH2O       Electrolysis water consumption
O2GENH2 Electrolysis hydrogen production
O2GENO2 Electrolysis oxygen production
EDCH2          EDC hydrogen consumption
CO2REMO2       total O2 consumed by CO2 removal process
EDCO2;EDC      oxygen consumption
CO2REMH2       total H2 consumed by CO2 removal process
CO2REM         total CO2 removed by CO2 removal process
CO2RMWAT       SAWD water consumption
CO2RMH2O       total H2O produced by CO2 removal processes
CO2REDH2       total H2 consumed by CO2 reduction processes
CO2RDH2O       total H2O produced by CO2 reduction processes
CO2RDCH4       total CH4 produced by CO2 reduction processes
CO2RED         total CO2 reduced by CO2 reduction processes
BOSCHH2 BOSCH H2 consumption
CO2REDC BOSCH solid carbon produced
WASHH2O WASH H2O consumption
WASHLOAD       WASH latent H2O load
FLUSH1         PREWAST H2O consumption
URINMIX PREWAST total mixture output
COND           condensate
```

```
STMASS          initial STORE mass
XMASS1          final STORE mass
DMASS           change in STORE mass
```

APPENDIX C. THE MODEL (.MOD) FILE

All information necessary to create the subsystem screens is contained in the MOD file. Since this file contains graphical information, it is highly recommended that this file not be modified through means other than CASE/A (i.e. using the VAX editor). The MOD file is strictly formatted and contains a considerable amount of unlabeled data. As such, it is extremely difficult to debug a corrupted MOD file. To illustrate the MOD file, an example example schematic is shown in figure C-1. The associated MOD file is shown in figure C-2 and described below.

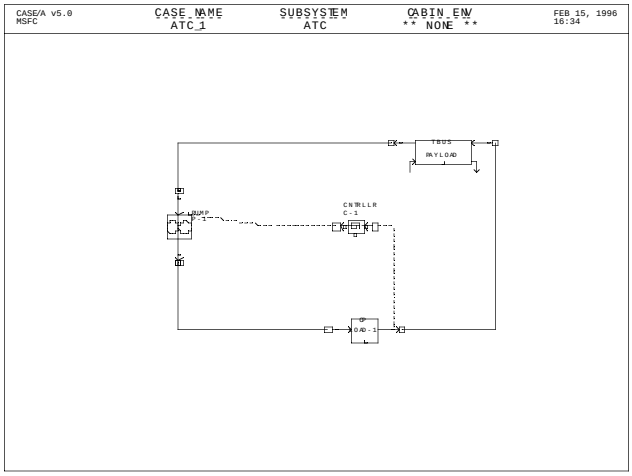


Figure C-1. Example subsystem schematic.

```
  4   5   0   0   0   0   0   0    24    19    22    15    15   121
ATC      27 P-1       301  426    0 270     5     0     0     1
ATC      32 LOAD-1    651  206    0   0     3     0     0     2
ATC      33 PAYLOAD   650  554    0  90     3     0     0     3
ATC      50 C-1       607  420    0   0     3     0     0     4
   1    2    2    1    0    2  301  206    0    0    0    0    0    0    0    0
   1  600
   2   -2    4    2    0    2  698  420    0    0    0    0    0    0    0    0
   1  600
   4    1    1   99    0    3  452  420  452  446    0    0    0    0    0    0
   1  600
   2    2    3    1    0    2  736  206    0    0    0    0    0    0    0    0
   1  600
   3    2    1    1    0    2  301  570    0    0    0    0    0    0    0    0
   1  600
```

Figure C-2. Example model description file (.MOD FILE).

Model File Description:

Line 1—This header line contains general information about the model. There are 14 items as follows:

Item	Description	Example
1)	number of components in model	(4)
2)	number of connections	(5)
3)	number of labels	(0)
4)	number of subsystems assigned to cabins	(0)
5)	number of cabins with assigned subsystems	(0)
6)	number of SPLIT components	(0)
7)	number of BLACKBOX components	(0)
8)	number of notes	(0)
9)	record number of LABELS data base	(24)
10)	record number of CONTROL data base	(19)
11)	record number of PLOT data base	(22)
12)	record number of USERCON data base	(15)
13)	record number of ARCHIVE data base	(15)
14)	not currently in use	(121)

Note that the remaining lines in this file are grouped according to items (1) through (8) above. In other words, the components are listed first, then all of the connections, followed by labels, etc.

Lines 2 to 5—These lines are a listing of all components in the model. Initially, the order of the components depends upon the order in which they were located in the model. After a solution has been invoked, however, this list is resequenced according to the component priority code discussed in section 4.1.2. The ten items on each line of this section are described below (examples are for the first line only).

Item	Description	Example
1)	subsystem name (8 characters)	(ATC)
2)	equipment type (ITYPE)	(27)
3)	component name (8 characters)	(P-1)
4)	x-coordinate (0,0 at lower left of screen)	(301)
5)	y-coordinate (0,0 at lower left of screen)	(426)
6)	cabin assignment info (by bits)	(0)

7)	orientation angle (IROT)	(270)
8)	record number in respective data base	(5)
9)	unused (0)	
10)	unused (0)	
11)	equipment number	(1)

Lines 6 to 10—These lines contain connection information. The 18 items are described below:

Item	Description	Example
1)	component “A”	(1)
2)	stream “A”	(2)
3)	component “B”	(2)
4)	stream “B”	(1)
5)	unused (0)	
6)	number of segments	(2)
7, 8)	(x,y) of intermediate point 1	(301) (206)
9, 10)	(x,y) of intermediate point 2	(0) (0)
11, 12)	(x,y) of intermediate point 3 (0) (0)	
13, 14)	(x,y) of intermediate point 4 (0) (0)	
15, 16)	(x,y) of intermediate point 5 (0) (0)	
17)	equivalent length (feet)	(1)
18)	equivalent diameter (hundredths of inch)	(600)

The next section of the MOD file is normally reserved for stream label data. Recall from the header line, however, that there are no lables in this model. One can determine the details of this section by examination of LOADCASE.FOR and using the appendix B, common block variables, as a reference.

The remaining section of the .MOD file can contain BLACKBOX component data and NOTE data neither of which are present in this model. There are three types of notes: C is a note centered at (x,y) containing 4 strings 20 characters long; L is a note left justified at (x,y) containing 4 strings of 20 characters, and N is a note left justified at (x,y) containing 1 string of 80 characters. The format for the NOTE information:

Item	Description
1)	subsystem name
2)	note type
3)	character size, 1-4 (4 is smaller)
4)	string(s), total of 80 characters
5)	x-coordinate
6)	y-coordinate

APPENDIX D. INDEX OF CASE/A SUBROUTINES

This appendix contains a listing of all CASE/A FORTRAN subroutines and the pages of this manual where descriptions of the routines can be found.

ROUTINE	PAGE
CASEAMAIN	37
Function DEWPT(PARTIAL_PRESS)	56
Function ROWATER(TEMP)	59
Function SATPR(TEMP)	81
Subroutine ADSORPTN	84
Subroutine AFSPE	85
Subroutine ARCHFILE	61
Subroutine ARCHIVE	61
Subroutine ASSIGN	53
Subroutine BCK_GD_CLR(IBACKGD)	17
Subroutine BDN (IX, IY, K, IROT,ISTM)	13
Subroutine BDNSHT (IX, IY, K, IROT,ISTM)	13
Subroutine BENCH(M,X)	55
Subroutine BIVAR(X,Y,A,Z)	55
Subroutine BLF (IX, IY, K, IROT,ISTM)	14
Subroutine BLFSHT (IX, IY, K, IROT,ISTM)	14
Subroutine BLOCK (IXSIZ, IYSIZ, IX, IY, IROT)	14
Subroutine BMR	85
Subroutine BOSCH	85
Subroutine BRT (IX, IY, K, IROT,ISTM)	14
Subroutine BRTSHT (IX, IY, K, IROT,ISTM)	14
Subroutine BUP (IX, IY, K, IROT,ISTM)	14
Subroutine BUPSHT (IX, IY, K, IROT,ISTM)	15
Subroutine CABIN	85
Subroutine CAP	85
Subroutine CDCODE (CLINEJ, LOC, ITYP, IWORD, IDATA, IERR)	28
Subroutine CDEL(NEA,NEB,ISTR)	55
Subroutine CDELA(NEA,NEB,ISTR)	55
Subroutine CEDIT(IEDIT)	27
Subroutine CFIELD (CLINEJ, LOC, ICOLL, ICOLH, IERR)	28
Subroutine CFR	86
Subroutine CHX	86
Subroutine CILLCHAR(CISC,LEN,CHRFLG)	55
Subroutine CIRCLE (IRAD, IX, IY)	15
Subroutine CLDRAI (NDATE, NYR, NMO, NDA)	28
Subroutine CLDRIA (NYR, NMO, NDA, NDATE)	28
Subroutine CLOADCASE(NAME)	7, 25
Subroutine CMPOPEN	25
Subroutine CNEWCASE(NAME)	8
Subroutine CNHX	86
Subroutine CNTRLLR	86
Subroutine COLOR(INDEX)	17
Subroutine COMPDP(NSTM,NSI,NSO,DP)	80
Subroutine CONDP(NEA,NSA,NEB,NSB,NC)	79
Subroutine CONECT	16
Subroutine CONINIT	55
Subroutine COPY(NUMSTM,CFLAG)	77
Subroutine COPYA(NUMSTM)	77
Subroutine COPYALL	9
Subroutine CORRECT(NC)	56

ROUTINE **PAGE**

Subroutine CP 87
Subroutine CREADAL (CLINEJ, ICOLL, ICOLR, IDATA, IERR) 28
Subroutine CREADALC (CLINEJ, ICOLL, ICOLR, IWORD, CDATA, IERR) 28
Subroutine CREADFL (CLINEJ, ICOLL, ICOLR, DATA, IERR) 29
Subroutine CREADIN (CLINEJ, ICOLL, ICOLR, IDATA, IERR) 29
Subroutine CREW 87
Subroutine CTOLOWERC(CCHAR,NCHAR) 56
Subroutine CTOUPPERC(CCHAR,NCHAR) 56
Subroutine CUTALL 9
Subroutine DCLFOR 52
Subroutine DEFLOW 87
Subroutine DEHUM 87
Subroutine DEL(IREC) 56
Subroutine DELCAS 7
Subroutine DELCN 16
Subroutine DELEQ(ICUT) 9
Subroutine DELLAB(ICUT) 9
Subroutine DELNOTE(ICUT) 10
Subroutine DELREC (IRECL, IRECH) 29
Subroutine DENVIS(NEA,NSA,NEB,NSB,DEN,VIS) 56
Subroutine DINTER(X,A,Y) 56
Subroutine DIR 7
Subroutine DIST (IX1, IY1, IX2, IY2, XD) 56
Subroutine DPCS(IEQA,ISTA,IEQB,ISTB,NCPT) 38
Subroutine DRAWC (KEQ) 15
Subroutine DRLABL (KLAB, MEQ) 10
Subroutine DRNOTE (K) 10
Subroutine DUPLICATE(IREC,JREC) 57
Subroutine EDC 87
Subroutine EDT 53
Subroutine EQLOAD 26
Subroutine EQOPEN (ITYPE) 26
Subroutine EQSOLVE 38
Subroutine ERRDUMP(ITEST) 57
Subroutine EVAP 88
Subroutine FIELD (ICOM, LOC, ICOLL, ICOLH, IERR) 29
Subroutine FILREC (IA, IDATA) 29
Subroutine FILTER 88
Subroutine FINDC(IEQ) 57
Subroutine FINDRM (IREC) 29
Subroutine FLAG 52
Subroutine FLOLEG(XM,DEN,VIS,XL,XD,XK,PDEL,MAX) 57
Subroutine FRAME1 17
Subroutine FRICTDP(SDEN,SVIS,SXL,SXD,XMDOT,DPX) 79
Subroutine GETC(NAME,ISTR,ICONST,VALUE) 73
Subroutine GETI(NAME,IEQVAL) 73
Subroutine GETK(NAME,ICON,VALUE) 73
Subroutine GETP(NAME,ISTR,VALUE) 73
Subroutine GETT(NAME,ISTR,VALUE) 74
Subroutine GETU(ILOC,VALUE) 74
Subroutine GIMAG(IEQB,ISTB,NCPT,PDEL,MAX,GIX) 57
Subroutine GRALPH 18
Subroutine GRCHRZ (ISIZ) 18
Subroutine GRCOPY 18
Subroutine GRCUSR (ICHAR, IX, IY) 18
Subroutine GRDRAW (IX, IY) 18

Subroutine GRERAS 18
Subroutine GRINIT 18
Subroutine GRLBAB (IX, IY, ISTRING, LSTRING) 18
Subroutine GRLBCT (IX, IY, ISTRING, LSTRING) 18
Subroutine GRMOVE (IX, IY) 18
Subroutine GRSCREEN(IWIDE) 19
Subroutine H2OSEP 88
Subroutine HATCH 88
Subroutine HEADER 41
Subroutine HEATER 88
Subroutine HELP 52
Subroutine HIT (IX, IY, JEQ, NSTR,IXC,IYC) 17
Subroutine HITBOX(IX,IY,IXOFF,IYOFF,ISTM) 15
Subroutine HX 89
Subroutine INTER(X,A,Y) 57
Subroutine IONEXCH 89
Subroutine ITEMIO (ITEM, IA) 30
Subroutine ITMOUT(ITEM,IA,IOUT) 26
Subroutine KAM2AS (NCHAR, KA4, KADE) 30
Subroutine KAS2AM (NCHAR, KADE, KA4) 30
Subroutine KHECK(NUSTM) 57
Subroutine KHECKA(NUSTM) 58
Subroutine LABL 10
Subroutine LBIT (NS, NL, IW, IV) 30
Subroutine LIOH 89
Subroutine LISOPEN 26
Subroutine LOADCOND(IEQA,ISTA,IEQB,ISTB,XM,DP) 58
Subroutine LOCATE 10
Subroutine LTSEMCIR (IRAD, IX, IY, IOFF, IROT) 15
Subroutine MASSFRAC(NUM,YI,XMWI,XI) 58
Subroutine MERGE 54
Subroutine MNETWK 81
Subroutine MODBAK 58
Subroutine MODULE 89
Subroutine MOLEFRAC(NUM,PRESSI,YI) 58
Subroutine MOLSIEV 89
Subroutine MOVALL 11
Subroutine MOVEIT 11
Subroutine MOVLAB 11
Subroutine MOVNOTE 11
Subroutine MSPLT 90
Subroutine MVBITS (ISORC, ISTRT1, ILEN, IDEST, ISTRT2) 30
Subroutine NODE 90
Subroutine NOTE 12
Subroutine O2N2 90
Subroutine OPS0 67
Subroutine OPS1 67
Subroutine OPS2 67
Subroutine OPS3 67
Subroutine OPS4 67
Subroutine OPS5 68
Subroutine OPS6 68
Subroutine OPS7 ..... 68, 90
Subroutine PASSIVE(NSTM,NSI,NSO) 58
Subroutine PDEL(NS) 58
Subroutine PIPE 90

ROUTINE **PAGE**

Subroutine PIPEDP(NSTM,NSI,NSO,XL,D) 80
Subroutine PLTDATA(NUM,PLD_ERR) 43
Subroutine PLTFILE(IA,PLF_ERR) 43
Subroutine POINTCON 58
Subroutine PREPRO 59, 74
Subroutine PREWAST 91
Subroutine PRNTSS 41
Subroutine PROPS(NE,NS,NC) 81
Subroutine PSEUDO 39
Subroutine PSPEC(NSTR,PRESS) 80
Subroutine PTMOD 26
Subroutine PULLSTX 59
Subroutine PUMP 91
Subroutine PUSHSTX 59
Subroutine PWVSUM 41
Subroutine QEXCHG(IQ,FAE,GCONV,GCOND,TRAD,TCONV,TCOND) 59
Subroutine RACK 91
Subroutine RAD 91
Subroutine RANDIN(ILOCK,IUNIT,IREC,IA,NWD) 26
Subroutine RBIVAR(X,Y,A,Z) 59
Subroutine RBYTE (IBYTE, IVAL, IARY) 31
Subroutine READAL (ICOM, ICOLL, ICOLH, IWORDS, IDATA, IERR) 31
Subroutine REDRAW(JFLAG) 12
Subroutine RESET1 (IREC) 19
Subroutine RESTOR (IRECL,IRECH) 31
Subroutine RETRIEVE 61
Subroutine RINTER(Y,A,X) 59
Subroutine RNSS 12
Subroutine RO 91
Subroutine ROTATE 12
Subroutine RTSEMCIR (IRAD, IX, IY, IOFF, IROT) 16
Subroutine SABAT 92
Subroutine SAVE 8, 27
Subroutine SAVEAS(NAME) 8, 60
Subroutine SAWD 92
Subroutine SBYTE (IBYTE, IVAL, IARY) 31
Subroutine SCALE(IST) 60
Subroutine SCALER(ISTRM,NSTRM,DESFLW,ALWFLOW) 60
Subroutine SCREDT(IEDIT) 27
Subroutine SEMIRECT(IH,IW,IX,IY,IROT) 16
Subroutine SEQUENCE(NAME) 39
Subroutine SETC(NAME,ISTR,ICONST,VALUE) 74
Subroutine SETK(NAME,ICON,VALUE) 74
Subroutine SETP(NAME,ISTR,VALUE) 74
Subroutine SETT(NAME,ISTR,VALUE) 74
Subroutine SETU(ILOC,VALUE) 74
Subroutine SFWE 92
Subroutine SINK 92
Subroutine SMVBITS (IVAL1, ISTART, ILEN, IVAL2 ITO) 31
Subroutine SOLVE 39
Subroutine SORTIEQ 40
Subroutine SOURCE 92
Subroutine SSOUT(NA) 42
Subroutine SUBSYS 13
Subroutine SUM 92
Subroutine SUMINIT 60

ROUTINE **PAGE**

Subroutine SUMMARY 42
Subroutine SYSBAL 76
Subroutine SYSCLK (NDATE, NTIME) 32
Subroutine TANK 93
Subroutine TBOUND(IQ,TRAD,TCONV,TCOND) 60
Subroutine TBUS 93
Subroutine TDEL(NS) 60
Subroutine TDELA(NS) 60
Subroutine TEK_ADV 19
Subroutine TEK_HOM 19
Subroutine TIMESC 93
Subroutine TIMER 93
Subroutine TIMESTEP(TSTEP,N,GSUM,CAPAC,TAU,MINILPS) 61
Subroutine TNETWK(MAX, RELAX, TSTEP, G1, G2, G3, G4, G5, FAE, G7, G8, CAP,
QMASS, QSHEL, TRAD, TCONV, TCOND, TIN, TMI, TOUT, TWAL,
TMAS, TSHEL, IC) 80
Subroutine TNETWK2(MAX, RELAX, TSTEP, G1, G2, G3, G4, G5, FAE, G7, G8, CAP,
QMASS, QSHEL, TRAD, TCONV, TCOND, TIN, TMI, TOUT, TWAL,
TMAS, TSHEL, IC) 80
Subroutine TRANSBOD (JEQ, IXC, IYC) 19
Subroutine TRANSCON(JEQ, NSTR, IXC, IYC) 19
Subroutine TRANSLT (IX, IY, IROT, IARRAY, JARRAY) 19
Subroutine TRANSLT1(JEQ,KEQ) 20
Subroutine TSTEP(A,B,C,D) 61
Subroutine TYCON (IA, BI) 32
Subroutine UNASSIGN 54
Subroutine VALVE 93
Subroutine VCD 94
Subroutine VISC(C,W,T0,XMU0,T,XMU) 81
Subroutine WASH 94
Subroutine WQM 94
Subroutine WRITCON 42